Departamento de Informática

PROYECTO FIN DE GRADO

Diseño y evaluación de un complemento de refactorización de C++ secuencial a Intel TBB para el entorno Eclipse

GRADO EN INGENIERÍA INFORMÁTICA
Especialidad en Ingeniería de Computadores

Autor: Sergio García-Blanes Ingelmo
Tutor: José Daniel García Sánchez

Colmenarejo, Septiembre 2012

Agradecimientos

Quiero dar las gracias a mi tutor José Daniel por darme la oportunidad de desarrollar este proyecto y por todo el tiempo que me ha dedicado con el fin de ayudarme para la realización del mismo.

A Tania por estar siempre a mi lado y motivarme con el proyecto.

Resumen

La refactorización de código es una técnica que permite realizar transformaciones automáticas y semiautomáticas de código fuente para mejorar la calidad de los programas.

Esta técnica puede usarse para ayudar a la paralelización de código C++ usando la biblioteca Intel TBB.

En este trabajo se ha desarrollado un plugin para el entorno Eclipse que permite realizar un conjunto básico de refactorizaciones que ayuden a migrar aplicaciones escritas en C++ secuencial a Intel TBB.

Mediante este plugin el programador se encuentra a un solo clic de transformar y paralelizar un simple bucle for mediante la biblioteca Intel TBB. Esto supone un notable ahorro de tiempo a la hora de programar y una mayor velocidad de ejecución del código.

El documento proporciona una visión general acerca de la biblioteca Intel TBB y como es de gran utilidad para mejorar un código escrito en C++ secuencial. Además, se incluye en el documento como se ha llevado a cabo el desarrollo del complemento para Eclipse abarcando las fases de análisis, diseño e implementación.

Palabras clave: refactorización, transformaciones, paralelización, código, C++, TBB, for, parallel_for, plugin, Intel.

Índice

Índice de figuras

Índice de tablas

Índice de ilustraciones

Capítulo 1

Introducción

1.1 Motivación

La meta de cualquier programa es reducir al mínimo el tiempo total de cómputo del mismo. Para ello, una de las técnicas más efectivas es el paralelización del código, es decir, la realización de varias tareas compatibles al mismo tiempo.

El programador es capaz de reducir el tiempo de cómputo distribuyendo la carga de trabajo entre los procesadores disponibles. Además, la paralelización es un factor básico para tener un alto desempeño en los equipos de cómputo.

La velocidad de procesamiento no es solamente la razón para utilizar el paralelismo. La construcción de aplicaciones más complejas han requerido una computadora más rápida, y las limitaciones en el desarrollo adicional de computadoras seriales han llegado a ser más y más evidentes. [1]

Las estructuras de control de programación en las que la paralelización se centra son los bucles, ya que, en general, la mayor parte del tiempo de ejecución de un programa tiene lugar en el interior de algún bucle. Un compilador paralelo intenta dividir un bucle de forma que sus iteraciones puedan ser ejecutadas en microprocesadores separados de forma concurrente. [2]

Para facilitar la labor del programador existe una biblioteca de Intel denominada Threading Building Blocks (Intel TBB). TBB es una librería escrita en C++, para programas hechos en este lenguaje, que facilita la paralelización de los mismos en procesadores multicore gracias a las funciones y estructuras que provee. Su objetivo es lograr un paralelismo escalable, haciendo uso de C++ estándar, que además abstraiga al programador de los detalles de la plataforma y del manejo de hilos, que exprese el paralelismo de una manera más simple y que realice un mejor aprovechamiento de los recursos de los procesadores.

TBB permite expresar el paralelismo como tareas en lugar de hilos. De esta forma, no es necesaria llevar a cabo manualmente la gestión de los mismos: no son necesarias las sentencias create, join, manage, etc. Dichas tareas son asignadas automáticamente a hilos, haciendo un uso eficiente de los recursos del procesador.

Una característica curiosa es el uso de una técnica denominada task stealing: la librería se encarga de asignar/desasignar tareas entre procesadores de manera transparente al programador, según la carga de trabajo actual, para equilibrar el trabajo de todos. Toda esta “abstracción” permite desarrollar soluciones más simples de alto nivel.
Además, es compatible con otros paquetes de paralelización, por lo que se puede optar por usar algunos componentes de TBB bajo ciertas circunstancias, y usar diferentes soluciones (OpenMP, gestión de hilos manualmente) en otras. [3]

Con este proyecto se pretende que el uso de la biblioteca Intel TBB sea automático y transparente para el programador, gracias a la creación de un complemento de refactorización para el entorno de desarrollo Eclipse.

A continuación se describen en profundidad los objetivos que se pretende cumplir con este proyecto y se mostrará una breve estructura del presente documento.

1.2 Objetivos

El objetivo principal de este proyecto es el diseño, evaluación e implementación de un complemento de refactorización para el entorno de desarrollo Eclipse que permita a los programadores llevar a cabo una paralelización de un determinado código secuencial mediante la biblioteca Intel TBB. En este proyecto nos centraremos en la refactorización de cualquier bucle for que pueda ser paralelizado.

Por lo tanto, con este proyecto se pretende mejorar el tiempo de ejecución de la aplicación a desarrollar sin aumentar el tiempo de programación. Mediante este complemento se debe poder ofrecer la posibilidad de llevar a cabo la refactorización de una manera rápida y sencilla.

Para cumplir con los objetivos fundamentales del proyecto, el sistema deberá cumplir otra serie de objetivos:

- ✓ El sistema deberá mostrar al usuario cuando es posible la refactorización de una parte del código.
- ✓ El sistema mostrará una descripción de la refactorización antes de aceptar la misma.
- ✓ El sistema deberá autocompletar el código con las sentencias necesarias cuando se lleve a cabo la refactorización.
- ✓ El sistema deberá ofrecer al usuario desactivar el plugin por si este no desea utilizarlo.
- ✓ El sistema deberá ofrecer al usuario la posibilidad de modificar el número de iteraciones de los bucles for a partir de las cuales se permitirá la refactorización, despreciando el resto de bucles.

En este mismo documento se podrá seguir la evolución de los distintos objetivos y cómo se ha llevado a cabo su implementación.

1.3 Medios utilizados para el desarrollo del proyecto

Las siguientes herramientas han sido empleadas durante las distintas fases del desarrollo del proyecto:

- ✓ Se ha utilizado un PC con las siguientes características: Intel Core i5-2400 CPU 3,10 GHz, 4 núcleos y 8 Gb de memoria RAM.
- ✓ Sistema Operativo: Linux (Ubuntu 11.10) de 64 bits.
- ✓ Biblioteca Intel TBB 4.0.
- ✓ Entorno de desarrollo de plugin integrado de código abierto multiplataforma Eclipse (Eclipse PDE)
- ✓ Java Development Kit.
- ✓ Microsoft Office Project 2007 para la creación de los diagramas de Gantt.

1.4 Estructura del documento

La memoria del proyecto se ha dividido en distintos capítulos para explicar cada fase desarrollada en el proyecto:

El documento comienza por un repaso al contexto general del problema existente y que soluciones se han propuesto para dicho problemas. Además, se expone que se pretende conseguir con este proyecto y que medios han sido necesarios para conseguirlo.

En el segundo capítulo se pretende acercar al usuario a la tecnología utilizada (lenguaje de programación C++, biblioteca Intel TBB y entorno de desarrollo Eclipse) para mostrar una visión general del proyecto y el por qué de su realización. Se explican las características principales.

A continuación, comienza la planificación del proyecto junto con los estudios sobre la arquitectura utilizada, casos de uso y requisitos. Así mismo, con los datos extraídos anteriormente se determina el comportamiento esperado de la aplicación mediante una serie de diagramas de flujo del sistema y un prototipo de bajo nivel.

En el siguiente capítulo se ahonda en el desarrollo del sistema, es decir, como se ha conseguido llevar a cabo la implementación de las funcionalidades del plugin. Además incluye las pruebas realizadas para determinar el buen funcionamiento del mismo.

Más adelante, contiene el manual de usuario para el buen manejo del complemento de Eclipse, así como el manual de instalación para aquellos usuarios que no estén tan familiarizados con la entorno de desarrollo Eclipse.

El último capítulo nos muestra la sección de conclusiones, en la que se especifican las conclusiones generales, posibles trabajos futuros y se indica el presupuesto de la realización del proyecto.

Por último, se incluyen tres anexos en el presente documento:

Eclipse PDE, donde podemos encontrar una explicación sobre el entorno de desarrollo de plugin de Eclipse y cómo instalarlo correctamente.

Glosario, que incluye los términos utilizados en el desarrollo de la documentación para una mejor comprensión.

Referencias, que incluye toda documentación consultada para la elaboración del proyecto.

Capítulo 2

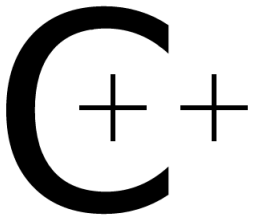
Estado del Arte

Con este capítulo se pretende acercar al usuario a la tecnología utilizada para la creación del proyecto. El complemento de refactorización se va a realizar para paralelizar código C++ secuencial mediante la biblioteca Intel TBB.

Por lo tanto, es necesario realizar un análisis del lenguaje de programación C++, de la biblioteca Intel TBB y el entorno de desarrollo para llevar a cabo dicho complemento.

Además, se va a proceder a comparar el método de paralelización usado con otra solución distinta existente en el mercado actual, OpenMP.

2.1 C++



C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. En un principio únicamente era una extensión del exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos y se denominó "C con clases". En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el lenguaje C++ es un lenguaje híbrido. [4] [5]

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

En la actualidad, C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. C++ mantiene las ventajas del lenguaje de programación C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del lenguaje de programación C original.

La evolución de C++ ha continuado con la aparición de JAVA, un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras.

Para terminar la explicación del lenguaje de programación C++, cabe destacar que C++ ha influido en algunos puntos muy importantes del ANSI C, como por ejemplo en la forma de declarar las funciones, en los punteros a void, etc. En efecto, aunque C++ es posterior a C, sus primeras versiones son anteriores al ANSI C, y algunas versiones de éste fueron tomadas de C++.

Como prueba de la gran aceptación del lenguaje de programación de C++ se puede hablar de la red social Facebook. La red social está desarrollada en PHP, pero debido al gran procesamiento que requerían los servidores de la aplicación, fue necesaria una transformación de PHP a C++. Para ello utilizaron un transformador de PHP a C++ llamado "Hip Hop for PHP". Esto no quiere decir que Facebook esté desarrollado en C++. La aplicación continúa estando desarrollada en PHP pero gracias a Hip Hop se transforma el código fuente PHP a código C++ optimizado y éste es compilado utilizando g++.

Gracias a la transformación, Facebook ahorra un 50% de procesamiento, lo que conlleva a menos consumo de energía, menos servidores y por tanto menos costes.

Este es un gran ejemplo de lo que se puede llegar a conseguir gracias en gran parte a C++.

En resumen, las ventajas de C++ frente a otros lenguajes son las siguientes:

- ✓ Lenguaje de programación orientado a objetos y multiparadigma.
- ✓ Lenguaje muy potente y robusto destinado a la creación de sistemas complejos.
- ✓ Es un lenguaje muy empleado, por lo que hay disponible un gran número de fuentes de documentación para solucionar cualquier problema que pueda surgir durante el desarrollo.

2.2 Eclipse C++

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para Visual Age. Eclipse es ahora desarrollado por la Fundación Eclipse,

una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse es un IDE conocido para el desarrollo de aplicaciones Java. Sin embargo, es un IDE mucho más flexible de lo imaginado, pues gracias a una infinidad de plugins permite, entra otras cosas, editar clases visuales de Java, programar aplicaciones J2EE, C/C++ y en varios lenguajes más, conectarse a bases de datos y escribir consultas SQL, etc.

Para esta ocasión, debemos utilizar el Eclipse C/C++ Development Tools (CDT). El proyecto CDT ofrece un completo y funcional entorno de desarrollo integrado para C/C++basado en la plataforma Eclipse. [6]

2.3 OpenMP

OpenMP es una interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join. [7]

Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Definido juntamente por un grupo de proveedores de hardware y de software mayores, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas para las plataformas que van desde las computadoras de escritorio hasta las supercomputadoras.

OpenMP se basa en el modelo fork-join, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join).

Cuando se incluye una directiva OpenMP esto implica que se incluye una sincronización obligatoria en todo el bloque. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que nos dé la directiva, y al final de ella habrá una barrera para la sincronización de los diferentes hilos (salvo que implícitamente se indique lo contrario con la directiva no wait). Este tipo de ejecución se denomina fork-join.

Existen diversas directivas, como por ejemplo:

- ✓ parallel: Esta directiva nos indica que la parte de código que la comprende puede ser ejecutada por varios hilos.
- ✓ for: Igual que parallel pero optimizado para los bucles for. Su formato es:

  #pragma parallelfor [cláusula, ... , cláusula]

2.3.1 Ventajas

Como se ha comentado anteriormente, se va a proceder a comparar y listar las ventajas de OpenMP con respecto a la solución de paralelización utilizada en el proyecto, Intel TBB [8]:

- ✓ OpenMP es más simple y tiene una curva de aprendizaje menor con respecto a Intel TBB.
- ✓ OpenMP permite con pocos cambios respecto al programa secuencial obtener versiones paralelas de buen rendimiento. Con TBB tendríamos que hacer mayores cambios para obtener los mismos resultados.
- ✓ OpenMP es un estándar abierto.
- ✓ TBB tiene un coste mayor en los bucles que OpenMP.

2.4 Threading Building Blocks (TBB)

2.4.1 Introducción

Intel Threading Building Blocks (Intel TBB) es una biblioteca en tiempo de ejecución basada en plantillas para C++ desarrollada por Intel para facilitar la escritura de programas que exploten las capacidades de paralelismo de los procesadores con arquitectura multinúcleo. [9] [10]

Esta biblioteca proporciona algoritmos y estructuras de datos que permiten al programador evitar en parte las complicaciones derivadas del uso de los paquetes nativos de gestión de hilos de ejecución en los que la creación, sincronización y

destrucción de los hilos es explícita y dependiente del sistema. En lugar de esto, la biblioteca abstrae el acceso a los múltiples procesadores permitiendo que las operaciones sean tratadas como tareas que se reparten automática y dinámicamente entre los procesadores disponibles mediante un gestor en tiempo de ejecución.

Esta aproximación hace que Intel TBB se incluya en la familia de soluciones para la programación paralela que permiten desacoplar la programación de las características particulares de la máquina.

Se trata por lo tanto de una librería de paralelización de propósito general para C++ en la que el programador define una serie de tareas paralelas que la propia librería se encarga de asignar a hilos de ejecución concurrentes de forma transparente. Esto permite al programador centrarse en la extracción de paralelismo de los algoritmos sin la distracción de los mecanismos de implementación subyacentes.

Las librerías de paralelización presentan una serie de ventajas respecto a otros métodos para la extracción de paralelismo: son más portables, no exigen a los programadores el aprendizaje de nuevos lenguajes y facilitan la reutilización de código legado.

2.4.2 Funcionamiento

Intel TBB implementa task stealing (robo de tareas) para balancear la carga de trabajo sobre los núcleos de procesamiento disponibles con el fin de incrementar el aprovechamiento de los núcleos y la escalabilidad de los programas. Inicialmente la carga de trabajo se divide uniformemente entre los núcleos de procesamiento disponibles.

Si alguno de ellos termina su trabajo mientras otro todavía tiene una carga significativa en su cola de tareas, el gestor de tareas reasigna parte de este trabajo al núcleo inactivo.

Esta capacidad de reasignación dinámica desacopla la programación de la máquina, permitiendo que las aplicaciones escritas usando esta biblioteca se escalen para usar todos los núcleos de procesamiento disponibles si no se produce ningún cambio en el código fuente o en los ejecutables.

Esto provoca que el balanceo de carga se gestione de forma dinámica y eficiente, liberando, por lo tanto, al usuario de la gestión de los hilos de ejecución.

También se minimiza la labor de sincronización ya que ésta es implícita en la mayoría de los casos. De esta forma, no sólo conseguimos una mayor sencillez y legibilidad en

el código, sino que se evitan posibles ineficiencias o errores encubiertos derivados de un mal uso de los bloqueos.

Intel TBB, siguiendo el ejemplo de la STL, está basada en el uso de plantillas ya que se espera que el polimorfismo en tiempo de compilación sea más eficiente que el tradicional polimorfismo en tiempo de ejecución. Además, de este modo se dota de generalidad y eficiencia a todas las herramientas que proporciona.

2.4.3 Contenidos

Intel TBB aporta, entre otros, la siguiente colección de componentes para la programación paralela:

- ✓ Algoritmos básicos: parallel_for, parallel_reduce, parallel_scan.
- ✓ Algoritmos avanzados: parallel_while, parallel_do, pipeline, parallel_sort.
- ✓ Contenedores: concurrent_queue, concurrent_vector, concurrent_hash_map.
- ✓ Reserva de memoria: scalable_allocator, cache_aligned_allocator.
- ✓ Exclusión mutua: mutex, spin_mutex, queuing_mutex, spin_rw_mutex, queuing_rw_mutex, recursive mutex.
- ✓ Operaciones atómicas: fetch_and_increment, fetch_and_decrement, fetch_and_add, compare_and_swap, fetch_and_store.
- ✓ Toma de tiempos: tick_count.
- ✓ Gestor de tareas: task.

2.4.4 Ventajas

A pesar de las ventajas de OpenMP, la mejor solución existente en el mercado para paralelizar un código C++ secuencial es la biblioteca Intel TBB:

- ✓ TBB es una librería; el compilador no necesita nada especial para compilar.
- ✓ TBB no requiere que el programador se preocupe por las políticas de planificación (static, dynamic y guided).

- ✓ TBB nos provee contenedores concurrentes (thread-safe).
- ✓ TBB encaja especialmente en el código que está altamente orientado a objetos (C++), y hace un uso extenso de plantillas y tipos de datos definidos.

Capítulo 3

Análisis y Diseño

3.1 Biblioteca Intel TBB: Parallel_for

Intel TBB aporta, entre otros, el algoritmo parallel_for para la programación en paralelo. La plantilla de función parallel_for nos permite romper un problema computacional expresado por un rango en subproblemas más pequeños que pueden ser procesados por separado. Su utilidad principal es la paralelización de bucles en los que las iteraciones son independientes entre sí. [11]

El algoritmo parallel_for particiona las tareas de una manera óptima para la ejecución en paralelo. Utiliza un algoritmo por el cual se produce un equilibrio de carga entre las distintas particiones. Cuando una iteración del bucle bloquea de forma cooperativa, el runtime redistribuye a otros subprocesos o procesadores el intervalo de iteraciones asignado al subproceso actual. Del mismo modo, cuando un subproceso completa un intervalo de iteraciones, el runtime también redistribuye el trabajo de otros subprocesos a ese subproceso.

Por otro lado, el algoritmo parallel_for también admite paralelismo anidado. Cuando un bucle paralelo contiene otro bucle paralelo, el runtime coordina los recursos de procesamiento entre los cuerpos de bucle de manera eficaz para la ejecución en paralelo.

El algoritmo del parallel_for tiene varias estructuras posibles, pudiendo tomar un valor inicial, un valor final y una función de trabajo (una expresión lambda, objeto de función o puntero a función), o tomar un valor inicial, un valor final, un valor de incremento y una función de trabajo. Pero la estructura que vamos a utilizar será la siguiente:

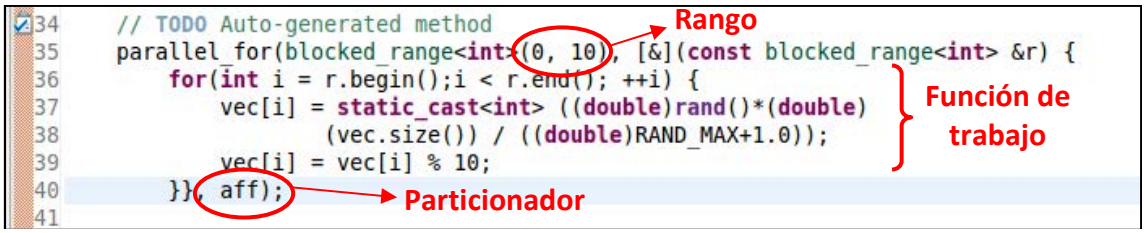


Ilustración 1. Componentes Parallel_for

Como podemos ver, utilizamos un valor inicial y un valor final para indicar el rango, una función de trabajo y el objeto "aff" (objeto del tipo affinity_particioner) que indica la política de particionado.

El algoritmo parallel_for se diferencia de la instrucción for en los aspectos siguientes:

- ✓ El algoritmo parallel_for no ejecuta las tareas en un orden predeterminado.

- ✓ El algoritmo parallel_for no admite las condiciones de finalización arbitrarias. Solo se detiene cuando el valor actual de la variable de iteración es uno menos que el valor final introducido en el rango.

- ✓ Los valores del rango deben de ser de tipo entero. Este tipo entero puede ser con signo o sin signo.

- ✓ La iteración del bucle debe ser hacia delante.

- ✓ El mecanismo de control de excepciones para el algoritmo parallel_for difiere del de un bucle for. Si se producen varias excepciones simultáneamente en el cuerpo de un bucle paralelo, el runtime propaga solo una de las excepciones al subproceso que llamó a parallel_for. Además, cuando una iteración del bucle produce una excepción, el runtime no detiene inmediatamente el bucle completo. En su lugar, el bucle se coloca en el estado cancelado y el runtime descarta cualquier tarea que no se haya iniciado.

Aunque el algoritmo parallel_for no admite condiciones de finalización arbitrarias, se puede usar la cancelación para detener todas las tareas.

3.2 Introducción al complemento de refactorización

Eclipse es una plataforma abierta y está diseñado para ser fácil e infinitamente extensible por terceros. En el centro del programa se encuentra el SDK de Eclipse, en torno al cual se pueden construir varios productos y herramientas. Así mismo, estos productos o herramientas pueden extenderse a su vez por otros productos y herramientas. Es decir, podemos extender un simple editor de texto para crear un editor de XML. Esta extensibilidad se puede lograr mediante la creación de plugins y su posterior uso. [12]

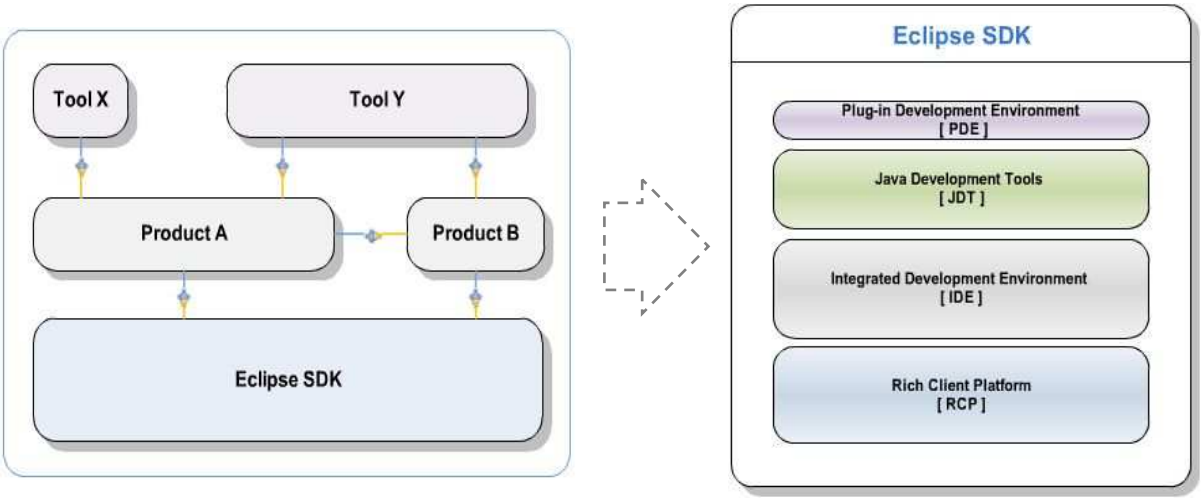


Figura 1. Componentes Eclipse SDK

Como se puede observar en las imágenes anteriores, una parte del SDK de Eclipse es el Plug-in Development Environment. Eclipse PDE, es la plataforma que proporciona todas las herramientas necesarias para desarrollar plugins y aplicaciones de RCP.

Todas las capas de Eclipse SDK se componen de plugins. Un plugin es una pequeña unidad de la plataforma Eclipse que se puede desarrollar por separado. Además, como vemos en la siguiente imagen, es un paquete autónomo que contiene el código y los recursos necesarios para funcionar.

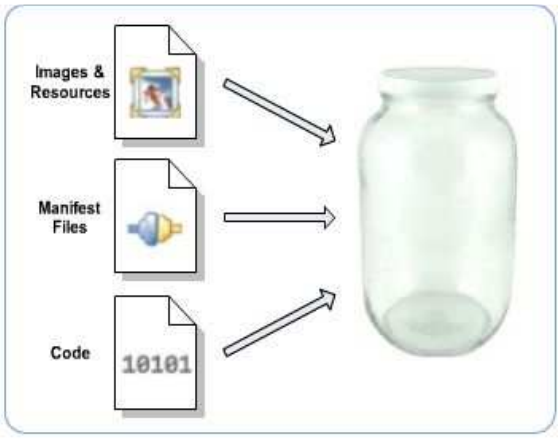


Figura 2. Contenido Plugin (I)

En este sentido, se va a desarrollar un plugin para el entorno de desarrollo de Eclipse C/C++. Este complemento se va a encargar de paralelizar código C++ secuencial, concretamente bucles for, utilizando la función parallel_for de la biblioteca Intel TBB (explicado en el punto 3.1).

Es decir, se pretende transformar los bucles for en parallel_for manteniendo el rango, el incremento y el contenido de la siguiente forma:

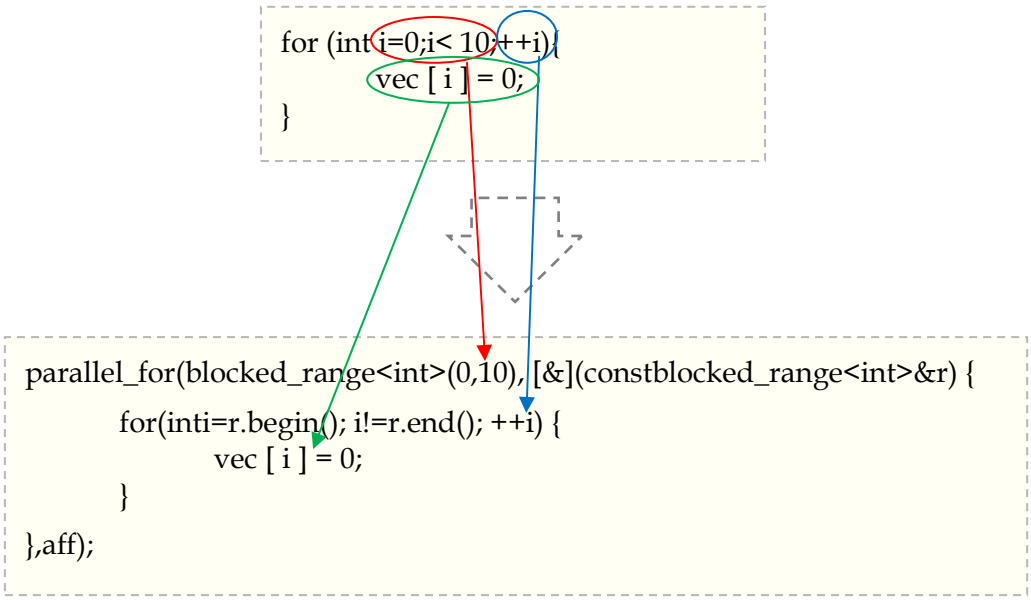


Ilustración 2. Convertir Bucle For en Parallel_for

Gracias a la creación de este plugin se pretende que el uso de la biblioteca Intel TBB sea automático y transparente para el programador, reduciendo con su uso el tiempo de cómputo del programa ya que se produce una distribución de la carga de trabajo entre los procesadores disponibles.

Inicialmente la carga de trabajo se divide uniformemente entre los núcleos de procesamiento disponibles. Si alguno de ellos termina su trabajo mientras otro todavía tiene una carga significativa en su cola de tareas, el gestor de tareas reasigna parte de este trabajo al núcleo inactivo.

Esta capacidad de reasignación dinámica desacopla la programación de la máquina, permitiendo que las aplicaciones escritas usando esta biblioteca se escalen para usar todos los núcleos de procesamiento disponibles sin ningún cambio en el código fuente o los ejecutables.

En definitiva, el complemento de refactorización debe ser capaz de realizar las siguientes acciones:

1. Detectar bucles for.

2. Comprobar si es posible refactorizar.

    a. Si es posible → Mostrar marca de warning para indicarlo.

    b. Si no es posible → No marcarlo de ninguna forma.

3. Transformar bucles for a la función parallel_for si el usuario lo desea.

Además, otra función muy importante que va a disponer el plugin es la función de autocompletar el código, es decir, el código está preparado para soportar un bucle for pero cuando llevas a cabo la refactorización el plugin va a analizar el código e inserta las sentencias que sean necesarias para compilar y ejecutar la función parallel_for.

En este sentido, son necesarios los includes "tbb/task_scheduler_init.h", "tbb/parallel_for.h", "tbb/partitioner.h" y "tbb/blocked_range.h", el namespace "tbb", el "affinity_particioner" y el "task_schedeuler_init".Si cualquiera de estas sentencias no se encuentran en el código, al realizar la refactorización aparecen de forma automática en el mismo.

Con este proyecto se pretende mejorar el tiempo de ejecución de la aplicación a desarrollar sin aumentar el tiempo de programación. Mediante este complemento se debe poder ofrecer la posibilidad de llevar a cabo la refactorización de una manera rápida y sencilla.

3.3 Planificación del proyecto

Para un correcto desarrollo del proyecto es necesario es necesario llevar a cabo una buena planificación del mismo.

El proyecto que se va a realizar es sencillo por lo que no se ha utilizado una metodología pesada, como métrica v3, para la documentación. Se ha pretendido enfatizar el producto en vez de la documentación, llevar a cabo un desarrollo iterativo, desarrollar únicamente lo que se necesita en cada momento y por tanto, que la documentación no sea demasiado robusta y lenta.

Por tanto, para un correcto desarrollo se va a dividir el trabajo en cuatro fases que se muestran de forma detallada a continuación:

Fases del proyecto			
Nombre	**Descripción**	**Operaciones**	**Duración**
Fase inicial	Análisis, diseño y desarrollo de las operaciones básicas del proyecto para que el mismo pueda funcionar.	Análisis Diseño Estructura del plugin Reconocimiento de bucles for Transformaciones básicas	6 semanas
Fase de desarrollo	Se llevan a cabo las condiciones exigidas por el cliente, se mejora la fase inicial, se introducen limitaciones y se prueba su correcto funcionamiento.	Mejora de las transformaciones Funciones autocompletar código Menú de preferencias Introducción de limitaciones Pruebas	6 semanas
Documentación	Se generan todos los documentos necesarios para la correcta gestión del proyecto.	Generación de gráficas Documentación	4 semanas
Fase final	Mejora de funciones y terminar los últimos detalles.	Pulir detalles Fin del proyecto	2 semanas

Tabla 1. Fases del proyecto

Para terminar, se muestra la planificación de las distintas fases, explicadas con anterioridad, que se llevaran a cabo para la realización del proyecto. Se muestra mediante un Diagrama de Gantt que podemos observar en la siguiente página:

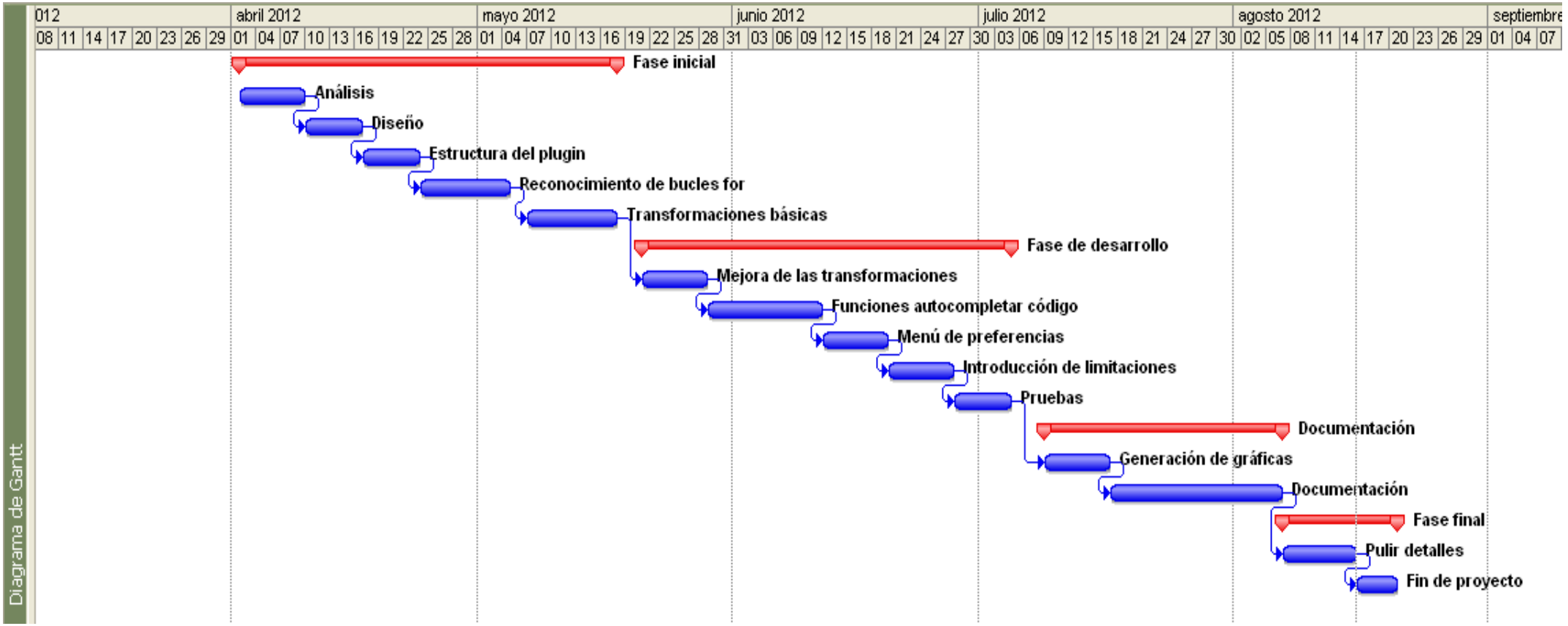


Figura 3. Diagrama de Gantt (Fases del proyecto)

3.4 Casos de uso

Un caso de uso es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. [13]

A lo largo de este apartado se especificarán los casos de uso para el complemento de refactorización que será desarrollado. Cada caso de uso se especificará mediante dos elementos:

- ✓ Diagramas para mostrar la interacción del usuario con el sistema.
- ✓ Tabla explicativa que describe el caso de uso.

Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

Por lo tanto, en este tipo de diagramas podemos destacar:

- ✓ Los casos de uso.
- ✓ El actor: es una entidad externa al sistema que se modela y que puede interactuar con él.
- ✓ Las relaciones: es una conexión entre los elementos del modelo, es decir, entre los casos de uso y actores. Pueden ser las siguientes: Un actor se comunica con un caso de uso, un caso de uso extiende otro caso de uso o un caso de uso usa otro caso de uso.

Por otro lado, la tabla explicativa tendrá el siguiente formato: [14]

- ✓ Nombre: identificador del caso de uso.
- ✓ Descripción: explicación del caso de uso.
- ✓ Actores: especifica un rol que adopta una entidad externa que interacciona directamente con el sistema.
- ✓ Precondiciones: condiciones que deben darse para la realización del caso de uso.

- ✓ Escenario básico: secuencia de acciones que describe la funcionalidad del caso de uso cuando no existen errores.
- ✓ Escenario alternativo: secuencia de acciones que describe la funcionalidad del caso de uso después de ocurrir un error.
- ✓ Postcondiciones: condiciones que son resultado de la ejecución del caso de uso.

A continuación se muestra de una forma gráfica todas las funcionalidades generales que abarcará el complemento de refactorización.

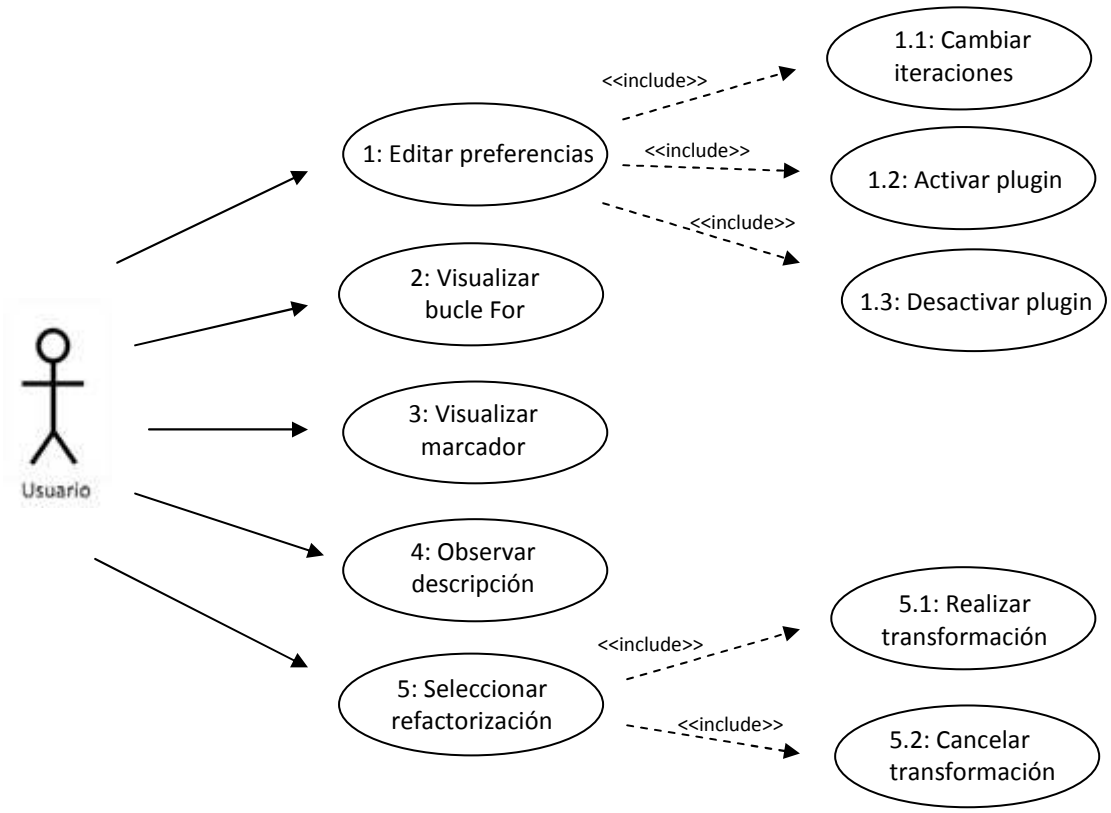


Figura 4. Casos de uso

Por otro lado, en las siguientes tablas explicativas se describe cada uno de los casos de uso representados en el diagrama:

CU1: Editar preferencias
Descripción: Permite al usuario acceder al menú de preferencias y visualizar las correspondientes propiedades del plugin que se va a desarrollar.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Flujo normal: *1.* Acceder al menú "Window" del menú horizontal situado en la parte de arriba del programa Eclipse. *2.* Pulsar el botón "Preferences". *3.* Pulsar sobre la opción "Plug-in Options".
Flujo Alternativo: *N / A*
Postcondiciones: Se podrán visualizar las propiedades del plugin.

Tabla 2. Caso de uso 1: Editar preferencias

CU1.1: Editar preferencias - Cambiar iteraciones
Descripción: Permite al usuario modificar el número de iteraciones a partir de los cuales los bucles for se pueden refactorizar.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Flujo normal: *1.* Acceder al menú "Window" del menú horizontal situado en la parte de arriba del programa Eclipse. *2.* Pulsar el botón "Preferences". *3.* Pulsar sobre la opción " Plug-in Options". *4.* Introducir un carácter numérico sobre el campo de texto "Numero de iteraciones". *5.* Pulsar el botón "OK".

Flujo Alternativo: *N / A*
Postcondiciones: Se modifica el número de iteraciones y se podrá refactorizar únicamente los bucles for con dichas iteraciones o más.

Tabla 3. Caso de Uso 1.1: Editar preferencias - Cambiar iteraciones

CU1.2: Editar preferencias- Activar plugin
Descripción: Permite al usuario activar el funcionamiento del plugin.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Flujo normal: 1. Acceder al menú "Window" del menú horizontal situado en la parte de arriba del programa Eclipse. 2. Pulsar el botón "Preferences". 3. Pulsar sobre la opción " Plug-in Options". 4. Seleccionar el radio button "Activar refactorizaciones". 5. Pulsar el botón "OK".
Flujo Alternativo: *N / A*
Postcondiciones: Se activa el plugin y se podrá refactorizar todos aquellos bucles for que sean permitidos.

Tabla 4. Caso de uso 1.2: Editar preferencias- Activar plugin

CU1.3: Editar preferencias- Desactivar plugin
Descripción: Permite al usuario desactivar el funcionamiento del plugin.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.

Flujo normal: 1. Acceder al menú "Window" del menú horizontal situado en la parte de arriba del programa Eclipse. 2. Pulsar el botón "Preferences". 3. Pulsar sobre la opción " Plug-in Options". 4. Seleccionar el radio button "Desactivar refactorizaciones". 5. Pulsar el botón "OK".
Flujo Alternativo: *N / A*
Postcondiciones: Se desactiva el plugin y NO se podrá refactorizar ningún bucle for.

Tabla 5. Caso de uso 1.3: Editar preferencias- Desactivar plugin

CU2: Visualizar bucle for
Descripción: Permite al usuario reconocer fácilmente los bucles for que pueden ser refactorizados subrayando el bucle for correspondiente.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Flujo normal: 1. Escribir un bucle for que cumpla las características para ser refactorizado. 2. Visualizar si puede ser refactorizado mediante el subrayado del bucle for.
Flujo Alternativo: *N / A*
Postcondiciones: El usuario observa que el bucle puede ser refactorizado y toma una decisión al respecto.

Tabla 6. Caso de uso 2: Visualizar bucle for

CU3: Visualizar marcador
Descripción: Permite al usuario reconocer fácilmente los bucles for que pueden ser refactorizados incluyendo una marca de Warning en la parte de la izquierda del editor de texto.

Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Flujo normal: *1.* Escribir un bucle for que cumpla las características para ser refactorizado. *2.* Visualizar si puede ser refactorizado mediante la marca de Warning.
Flujo Alternativo: *N / A*
Postcondiciones: El usuario observa que el bucle puede ser refactorizado y toma una decisión al respecto.

Tabla 7. Caso de uso 3: Visualizar marcador

CU4: Observar descripción
Descripción: Permite al usuario conocer la descripción de la transformación que se va a realizar antes de llevarla a cabo.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Escenario básico: *1.* Escribir un bucle for que cumpla las características para ser refactorizado. *2.* Pasar el ratón por encima del bucle que se va a transformar.
Escenario Alternativo: *1.* Escribir un bucle for que cumpla las características para ser refactorizado. *2.* Pulsar en el marcador de Warning situado en la parte izquierda del editor de Eclipse.
Postcondiciones: El usuario observa la descripción de la transformación que se va a realizar.

Tabla 8. Caso de uso 4: Observar descripción

CU5: Seleccionar refactorización
Descripción: Permite al usuario seleccionar un bucle for para poder llevar a cabo la transformación a la función "parallel_for" de la biblioteca Intel TBB.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin.
Escenario básico: 1. Escribir un bucle for que cumpla las características para ser refactorizado. 2. Pulsar en el marcador de Warning situado en la parte izquierda del editor de Eclipse.
Escenario Alternativo: N / A
Postcondiciones: El usuario observa un cuadro de diálogo en el que se le da la posibilidad de aceptar la transformación o cancelarla.

Tabla 9. Caso de uso 5: Seleccionar refactorización

CU5.1: Seleccionar refactorización - Realizar transformación
Descripción: Permite al usuario transformar un bucle for en la función "parallel_for" de la biblioteca Intel TBB.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin. ✓ Existir un bucle for que cumpla las características para ser refactorizado.
Escenario básico: 1. Pulsar en el marcador de Warning situado en la parte izquierda del editor de Eclipse. 2. Pulsar sobre el texto "Reemplazar con la función parallel_for".
Escenario Alternativo: N / A
Postcondiciones: El usuario observa que el bucle for se ha transformado y por tanto, paralelizado.

Tabla 10. Caso de uso 5.1: Seleccionar refactorización - Realizar transformación

CU5.2: Seleccionar refactorización - Cancelar transformación
Descripción: Permite al usuario cancelar la refactorización de un bucle for seleccionado.
Actores: Usuario
Pre-condiciones: ✓ Tener instalado Eclipse CDT. ✓ Tener instalado el plugin. ✓ Existir un bucle for que cumpla las características para ser refactorizado.
Flujo normal: 1. Pulsar en el marcador de Warning situado en la parte izquierda del editor de Eclipse. 2. Pulsar sobre otra parte del programa Eclipse distinta al cuadro de diálogo.
Flujo Alternativo: *N / A*
Postcondiciones: El usuario observa que el bucle for no se ha modificado.

Tabla 11. Caso de uso 5.2: Seleccionar refactorización - Cancelar transformación

3.5 Requisitos del sistema

En este apartado llevaremos a cabo la definición, análisis y validación de los requisitos a partir de la información facilitada por el cliente. El objetivo es obtener un catálogo de los requisitos donde se pueda comprobar que el producto generado se ajusta a dichos requisitos.

Los requisitos del sistema se pueden clasificar en dos tipos:

- ✓ Requisitos de capacidad (funcionales): Son aquellos requisitos que recogen un servicio requerido por el cliente/usuario. Describen el funcionamiento del sistema.

- ✓ Requisitos de restricción (no funcionales): Recogen algún tipo de limitación en la forma de prestar esos servicios o en la forma de desarrollarlos.

Definiremos cada requisito según el siguiente estándar:

- ✓ Identificador: Cada requisito de usuario deberá incluir un identificador para facilitar el seguimiento durante las fases posteriores.Debe respetarse el siguiente formato:

  RF-ZZ / RNF-ZZ

  Donde se identifica el tipo de requisito, es decir, funcional (RF) o no funcional (RNF). Mientras que ZZ es el número de secuencia que comienza para cada tipo en 01.

- ✓ Nombre: Representa de forma breve y descriptiva el requisito.

- ✓ Descripción: Describe el requisito de usuario. Debe ser breve pero intentando explicar de forma detallada la función del requisito.

- ✓ Necesidad**:** Los requisitos de usuario esenciales deben ser marcados como tal, no son prescindibles, otros pueden ser menos importantes y por tanto ser negociados. Tres niveles: esencial, conveniente u opcional.

- ✓ Prioridad**:** Cada requisito de usuario debe incluir una medida de prioridad para que el desarrollador pueda configurar el programa de trabajo.

- ✓ Estabilidad: Establece el valor de la estabilidad del requisito. La estabilidad puede variar a lo largo de la vida del producto. Los posibles valores que puede tomar la estabilidad son alta, media y baja.

- ✓ Verificabilidad**:** En cada requisito de usuario hay que indicar si es posible verificarlo o no; para ello se tienen que cumplir las siguientes condiciones: ha sido incorporado al diseño, se va a implementar en el software y ya ha sido comprobado su implementación en el software.

3.5.1 Requisitos funcionales

A continuación se muestran los requisitos funcionales (de capacidad) de nuestro sistema:

Identificador: RF-01
Nombre: Modificar propiedades
Descripción: El usuario podrá modificar las propiedades del plugin.
Necesidad: ☑Esencial☐Conveniente☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑Verificable ☐No verificable

Tabla 12. RF-01: Modificar propiedades

Identificador: RF-02
Nombre: Visualizar propiedades
Descripción: El usuario podrá visualizar las propiedades sin necesidad de aplicar ningún cambio.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 13. RF-02: Visualizar propiedades

Identificador: RF-03
Nombre: Almacenar propiedades
Descripción: Las propiedades deberán mantenerse en memoria al cerrar Eclipse y guardar los valores introducidos por el usuario para futuras sesiones.

Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 14. RF-03: Almacenar propiedades

Identificador: RF-04
Nombre: Iteraciones
Descripción: El usuario podrá modificar el número de iteraciones a partir de las cuales los bucles for podrán refactorizarse.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 15. RF-04: Iteraciones

Identificador: RF-05
Nombre: Desactivar
Descripción: El usuario podrá desactivar el plugin si lo cree conveniente.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja

Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 16. RF-05: Desactivar

Identificador: RF-06
Nombre: Activar
Descripción: El usuario podrá volver a activar el plugin en cualquier momento si este se encuentra desactivado.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 17. RF-06: Activar

Identificador: RF-07
Nombre: Visualizar marcador
Descripción: El usuario debe ser capaz de localizar un bucle for que se puede refactorizar mediante una marca de warning en el panel izquierdo del editor.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 18. RF-07: Visualizar marcador

Identificador: RF-08
Nombre: Marcar según iteraciones
Descripción: Se deben marcar los bucles for con más iteraciones de las indicadas en las propiedades.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 19. RF-08: Marcar según iteraciones

Identificador: RF-09
Nombre: Marcar si está activado
Descripción: Se deben marcar los bucles for cuando la opción "activar" se encuentra marcada en las propiedades.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 20. RF-09: Marcar si está activado

Identificador: RF-10
Nombre: Seleccionar marcador
Descripción: El usuario podrá seleccionar la marca de warning del bucle que desea paralelizar.

Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 21. RF-10: Seleccionar marcador

Identificador: RF-11
Nombre: Visualizar subrayado
Descripción: El usuario debe ser capaz de localizar un bucle for que se puede refactorizar mediante un subrayado del mismo en el panel del editor.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 22. RF-11: Visualizar subrayado

Identificador: RF-12
Nombre: Subrayar según iteraciones
Descripción: Se deben subrayar los bucles for con mas iteraciones de las indicadas en las propiedades.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja

Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 23. RF-12: Subrayar según iteraciones

Identificador: RF-13
Nombre: Subrayar si está activado
Descripción: Se deben subrayar los bucles for cuando la opción “activar” se encuentra marcada en las propiedades.
Necesidad: ☐ Esencial ☑ Conveniente ☐ Baja
Prioridad: ☐ Alta ☑ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 24. RF-13: Subrayar si está activado

Identificador: RF-14
Nombre: Descripción sobre subrayado
Descripción: El usuario podrá pasar el ratón por encima del bublefor subrayado y observar la descripción sin necesidad de hacer clic en el marcador.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 25. RF-14: Descripción sobre subrayado

Identificador: RF-15
Nombre: Descripción al refactorizar
Descripción: El usuario debe conocer la descripción del cambio que se va a producir al pinchar sobre el marcador pero antes de realizar la refactorización.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 26. RF-15: Descripción al refactorizar

Identificador: RF-16
Nombre: Cancelar transformación
Descripción: El usuario podrá cancelar la refactorización una vez a leído la descripción de la misma.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 27. RF-16: Cancelar transformación

Identificador: RF-17
Nombre: Aceptar transformación
Descripción: El usuario debe poder aceptar la refactorización de un bucle for.

Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 28. RF-17: Aceptar transformación

Identificador: RF-18
Nombre: Realizar transformación
Descripción: Al aceptar el usuario la refactorización, se debe convertir el bucle for seleccionado en la función parallel_for de la biblioteca Intel TBB.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 29. RF-18: Realizar transformación

Identificador: RF-19
Nombre: Comprobar transformación
Descripción: Al llevar a cabo la refactorización, se debe comprobar si se tiene en el código todas las sentencias necesarias para compilar y ejecutar la función parallel_for (namespaces, includes, …).
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja

Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 30. RF-19: Comprobar transformación

Identificador: RF-20
Nombre: Autocompletar código
Descripción: Autocompletar el código , si es necesario, con las sentencias necesarias para compilar y ejecutar la función parallel_for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 31. RF-20: Autocompletar código

3.5.2 Requisitos no funcionales

Así mismo, se muestran los requisitos no funcionales de nuestro sistema:

Identificador: RNF-01
Nombre: Números en las iteraciones
Descripción: Solo se puede incluir caracteres numéricos para indicar en las propiedades el número de iteraciones a partir de las cuáles se pueden refactorizar bucles for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja

Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 32. RNF-01: Números en las iteraciones

Identificador: RNF-02
Nombre: No vacío las iteraciones
Descripción: No se puede dejar vacío el campo de texto que sirve para indicar en las propiedades el número de iteraciones a partir de las cuáles se pueden refactorizar bucles for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 33. RNF-02: No vacío las iteraciones

Identificador: RNF-03
Nombre: *B*ucles for anidados marcador
Descripción: No se deben marcar como transformables los bucles for incluidos dentro de otros bucles for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 34. RNF-03: *B*ucles for anidados marcador

Identificador: RNF-04
Nombre: Bucles for + parallel_for + marcador
Descripción: No se deben marcar como transformables los bucles for incluidos dentro de la función parallel_for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 35. RNF-04: Bucles for + parallel_for + marcador

Identificador: RNF-05
Nombre: Bucles for anidados subrayado
Descripción: No se deben subrayar e indicar la posibilidad de refactorizar los bucles for incluidos dentro de otros bucles for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 36. RNF-05: Bucles for anidados subrayado

Identificador: RNF-06
Nombre: Bucles for + parallel_for + subrayado
Descripción: No se deben subrayar e indicar la posibilidad de refactorizar los bucles for incluidos dentro de la función parallel_for.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja

Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 37. RNF-06: Bucles for + parallel_for + subrayado

Identificador: RNF-07
Nombre: *B*ucles for decremento marcador
Descripción: No se deben marcar como transformables los bucles for con decremento.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 38. RNF-07: *B*ucles for decremento marcador

Identificador: RNF-08
Nombre: Bucles for decremento subrayar
Descripción: No se deben subrayar e indicar la posibilidad de refactorizar los bucles for con decremento.
Necesidad: ☑ Esencial ☐ Conveniente ☐ Baja
Prioridad: ☑ Alta ☐ Media ☐ Baja
Estabilidad: ☑ Alta ☐ Media ☐ Baja
Verificabilidad: ☑ Verificable ☐ No verificable

Tabla 39. RNF-08: Bucles for decremento subrayar

3.6 Matriz de trazabilidad

La matriz de trazabilidad permite comprobar a que requisito se corresponde cada caso de uso descrito con anterioridad. La matriz de trazabilidad de nuestro sistema es:

	CU1	CU1.1	CU1.2	CU1.3	CU2	CU3	CU4	CU5	CU5.1	CU5.2
RF1	✓									
RF2	✓									
RF3	✓									
RF4		✓								
RF5				✓						
RF6			✓							
RF7						✓				
RF8			✓			✓				
RF9				✓		✓				
RF10						✓	✓	✓		
RF11					✓					
RF12			✓		✓					
RF13				✓	✓					
RF14					✓		✓			
RF15								✓		
RF16							✓			✓
RF17									✓	
RF18									✓	
RF19									✓	
RF20									✓	
RNF1		✓								
RNF2		✓								
RNF3						✓				
RNF4						✓				
RNF5					✓					
RNF6					✓					
RNF7						✓				
RNF8					✓					

Tabla 40. Matriz de trazabilidad

3.7 Diagrama de flujo

Tras un análisis de los casos de uso y los requisitos, se procede a la creación del diagrama del flujo de la aplicación. [15]

El diagrama de flujo es la representación gráfica del proceso que se va a desarrollar. Estos diagramas utilizan símbolos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de fin de proceso.

Dicho diagrama se adecua a las necesidades del futuro sistema en conceptos del entorno de desarrollo y se dividirá en bloques cuyas funciones son realizadas por el usuario o por el sistema.

Gracias a los diagramas de flujo se puede comprender mucho mejor el sistema al mostrarlo como un dibujo y además permiten identificar los problemas y las oportunidades de mejora del proceso.

En este sentido podemos observar el diagrama de flujo de nuestro sistema:

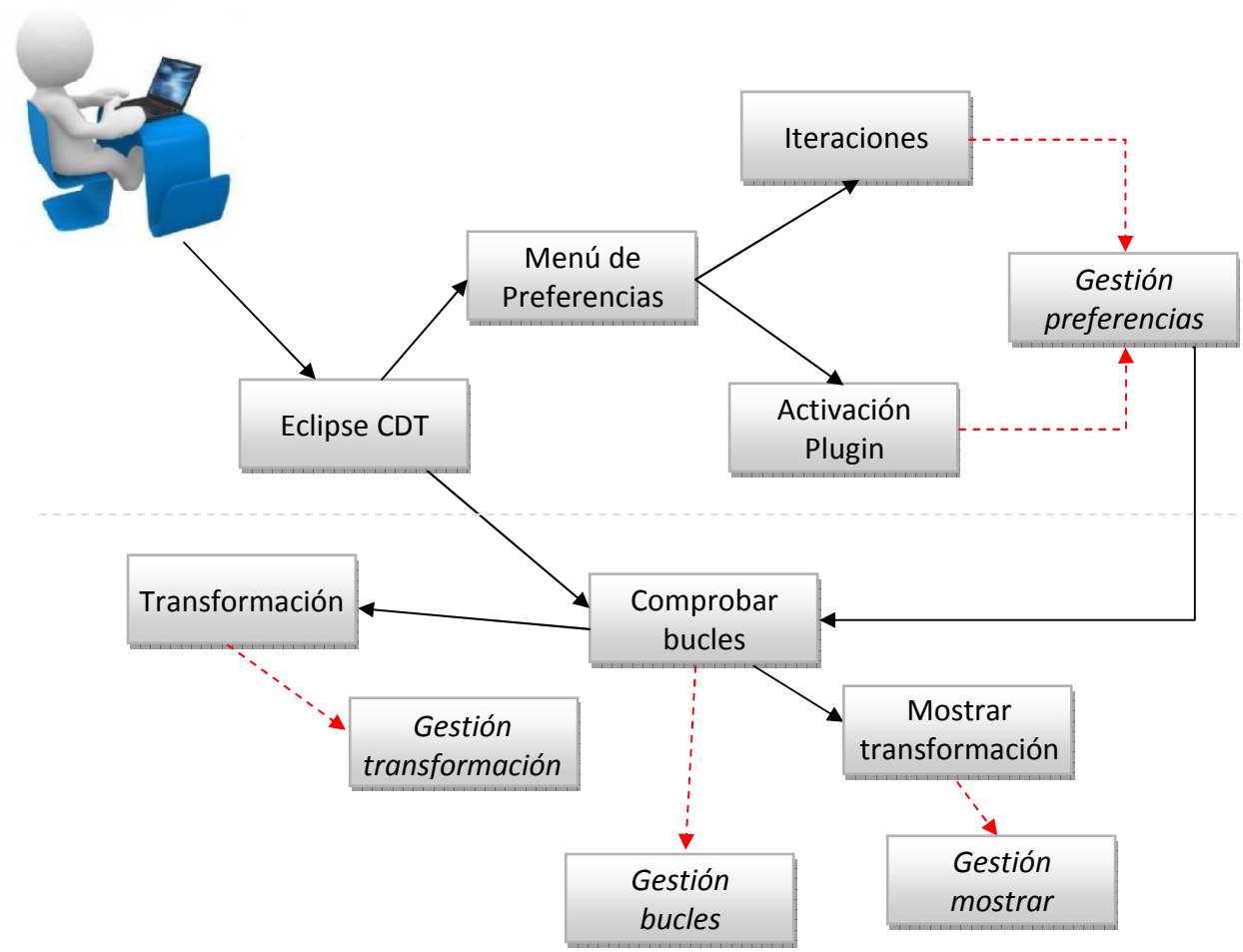


Figura 5. Diagrama de flujo

Comenzamos a observar que nuestro plugin va a disponer de dos partes claramente diferenciadas. Por un lado, el menú de preferencias y por otro, el complemento de refactorización propiamente dicho.

Un programador que tiene instalado el entorno de desarrollo Eclipse CDT y el complemento que estamos realizando, puede acceder al menú de preferencias del propio Eclipse y acceder a la nueva opción que hemos habilitado para modificar las propiedades del plugin. Se encuentra con dos opciones:

- ✓ El bloque de iteraciones: Se encarga de modificar el número de iteraciones a partir de las cuales los bucles for se pueden refactorizar.
- ✓ El bloque de activación del plugin: activa o desactiva el funcionamiento del plugin.

Además, existe un bloque de gestión de dichas preferencias, que se encarga de comunicar la decisión tomada por el usuario a la otra parte del plugin y además también se encargara de realizar operaciones internas como el almacenamiento de los valores introducidos para futuras sesiones.

Por otro lado, se encuentra el módulo del complemento de refactorización. En éste módulo todo gira en torno al bloque de comprobar bucles, el cual se encarga de analizar el código, obtener únicamente los bucles for e incluso descartar alguno de dichos bucles debido a las limitaciones impuestas por el cliente. Además, este bloque es el encargado de obtener las propiedades modificadas por el usuario en el menú de preferencias y descartar los bucles que no cumplan con dichas propiedades.

Por tanto, el bloque de gestión de bucles es el encargado de indicar al resto de bloques los bucles for que deben tratar. Estos otros bloques son los siguientes:

- ✓ El bloque de mostrar transformación: se encarga de obtener los bucles for enviados a través del bloque de gestión de bucles y mostrar al usuario un marcador y un subrayado de los bucles que pueden ser refactorizados. Además de este tratamiento, el bloque de gestión mostrar se encarga de mostrar la descripción de la transformación al pinchar sobre el marcador o pasar el ratón por encima del bucle for subrayado.

- ✓ El bloque de transformación: se encarga de llevar a cabo la transformación siempre y cuando el usuario acepte la misma. El bloque de gestión de transformación se encarga de modificar el código con la transformación indicada y autocompletar el mismo si fuese necesario para compilar y ejecutar la nueva función.

3.8 Diseño del complemento

En el anterior punto, se han mostrado los dos módulos que se van a desarrollar en el proyecto. Estos módulos tendrán su representación visual en el entorno de desarrollo de Eclipse, por lo que se va a realizar un prototipado de bajo nivel para mostrar el diseño que tendrán dichos módulos.

Si bien es cierto, que el diseño de la interfaz no depende de este proyecto ya que nosotros estamos desarrollando un plugin que se integrará en el entorno de desarrollo de Eclipse, acoplándose a su interfaz.

Por tanto, se va a mostrar en primer lugar como se puede reconocer nuestros módulos en Eclipse, en segundo lugar, el diseño del menú de propiedades del plugin y por último, el diseño de la pantalla de refactorización de Eclipse.

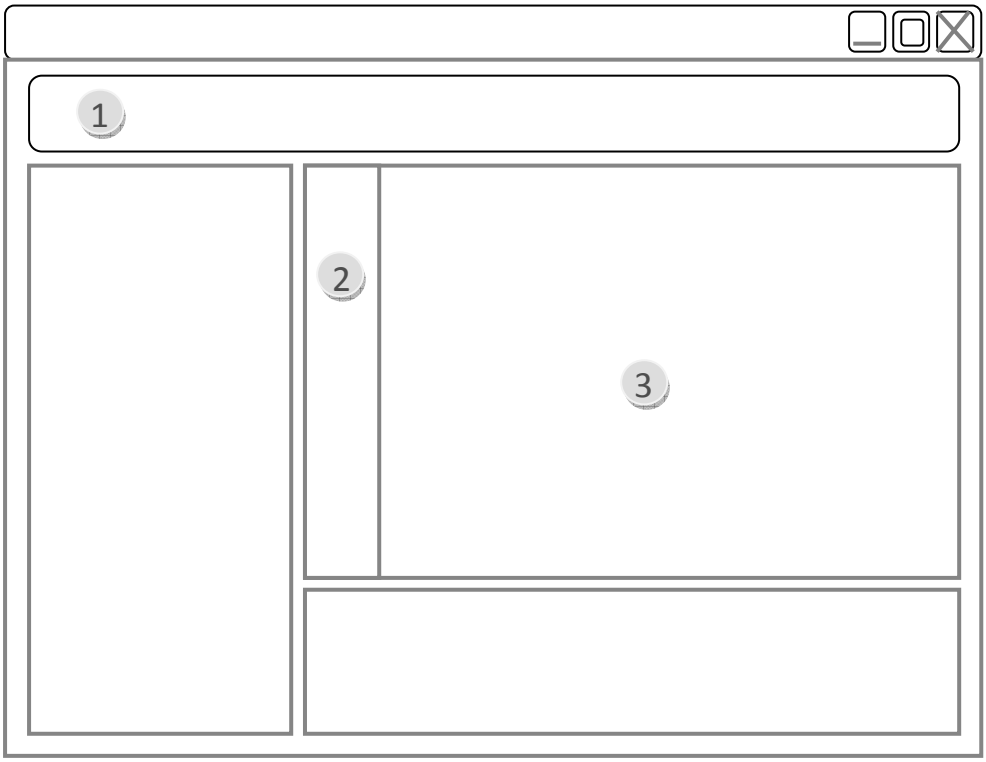


Figura 6. Diseño del complemento (I)

Como podemos observar en la imagen anterior, el entorno de desarrollo de Eclipse tiene varias partes diferenciadas claramente. Nosotros nos vamos a centrar en tres:

1. La primera es el menú horizontal de Eclipse. En el podemos ver el menú desplegable "Window" que mostrará a su vez la opción "Preferences". En este submenú se encuentra las propiedades del plugin que se va a desarrollar y que a continuación trataremos.

2. La segunda es la barra vertical que se encuentra al lado del editor de Eclipse. Es el espacio donde se mostrará el marcador para indicar que el bucle que se encuentra en esa línea puede ser refactorizado.

3. Y por último, nos encontramos con el editor de Eclipse. En este espacio es donde se inserta el código del programa que queremos crear. Si aparece un bucle for que puede ser refactorizado, en este espacio se subrayará dicho bucle.

Por otro lado, el menú de propiedades del plugin dispone como podemos ver de cuatro zonas:

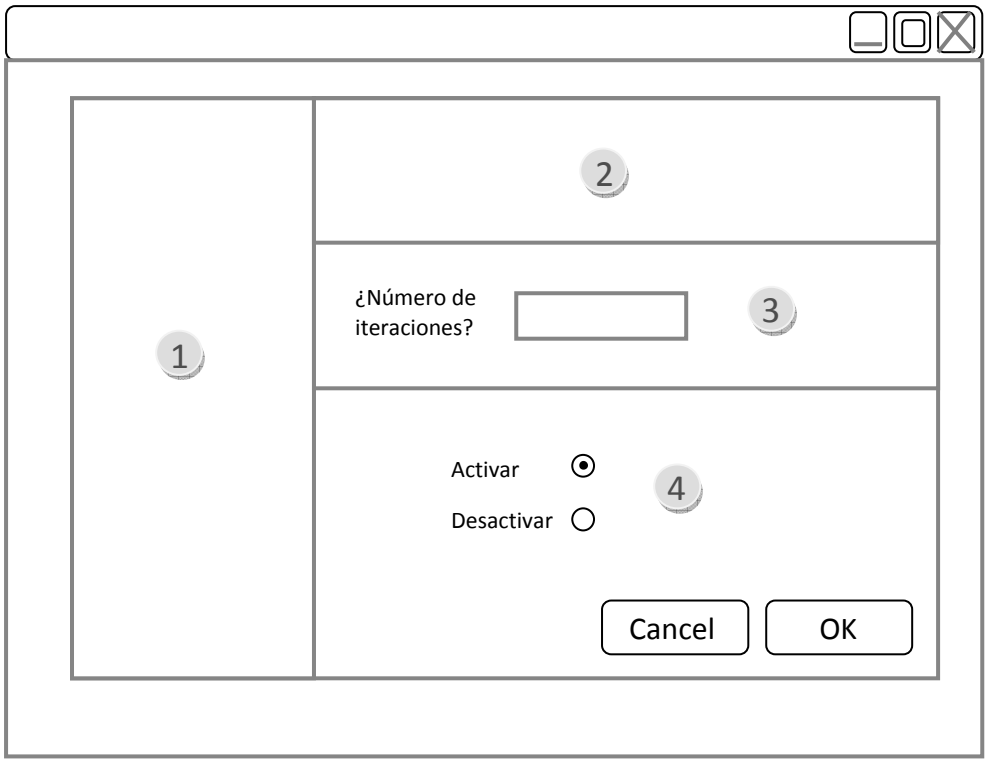


Figura 7. Diseño del complemento (II)

1. La primera es el menú vertical en el que se encuentran los distintos submenús de propiedades. Uno de ellos es el submenú de propiedades del plugin, que pinchando sobre él aparecen el resto de zonas.

2 La segunda, es un espacio reservado para explicar al usuario las propiedades que se pueden modificar y el por qué de las mismas.

3 La tercera zona es el espacio reservado para modificar el numero de iteraciones. Existirá un campo de texto para introducir el número de iteraciones. Solo permitirá caracteres numéricos.

4 La última, es la reservada para activar o desactivar el funcionamiento del complemento. Existirá una descripción y un radio button en el que sólo se podrá elegir una opción entre activar y desactivar.

Por último, la interfaz del cuadro de diálogo de las transformaciones no depende de nuestro proyecto pero es un módulo importante en el mismo y es necesario mostrar las partes que tendrá el mismo y como las vamos a personalizar:

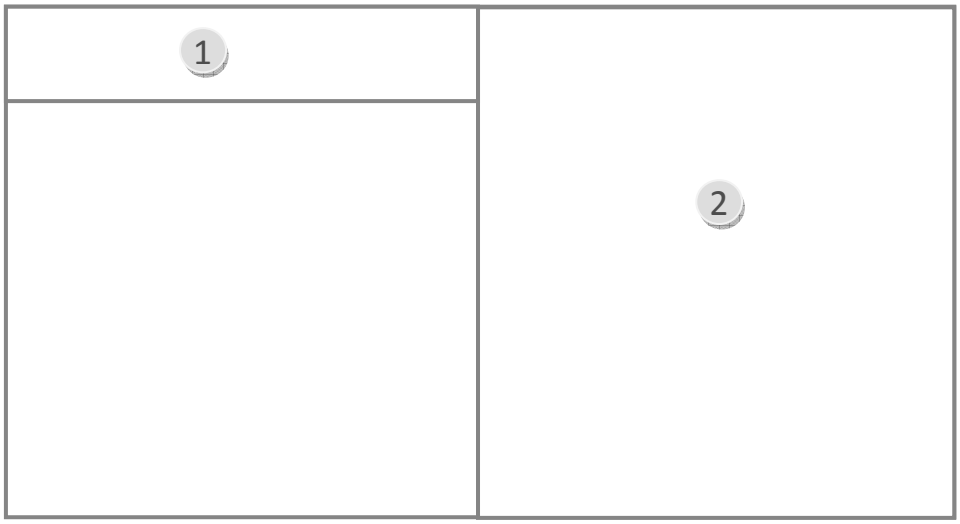


Figura 8. Diseño del complemento (III)

El cuadro de diálogo para las transformaciones tendrá dos partes diferenciadas que a continuación se explican:

1 La primera es la zona reservada donde se va a pinchar para aceptar la refactorización. El link para aceptar la transformación va a ser un texto.

2 La segunda, es un espacio reservado para la descripción del problema.

Capítulo 4

Implementación

Como hemos comentado con anterioridad un plugin es una pequeña unidad de la plataforma Eclipse que se puede desarrollar por separado. Es un paquete autónomo que contiene el código y los recursos necesarios para funcionar. [16]

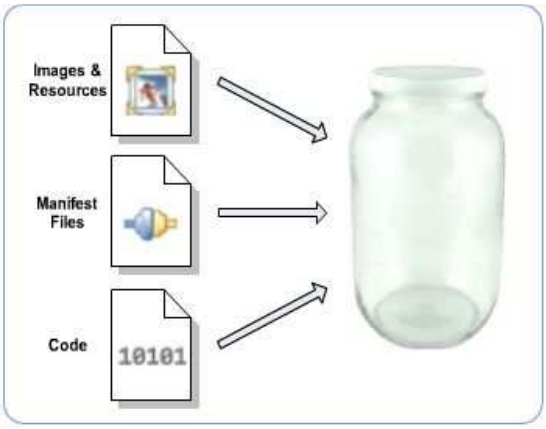


Figura 9. Contenido Plugin (II)

En este sentido, nuestro plugin está dividido en cuatro partes: el manifest file, el código para controlar la página de preferencias, el código para encontrar las posibles refactorizaciones y el código para llevar a cabo la transformación.

A continuación, pasaremos a explicar el desarrollo de cada uno de ellos de una forma detallada.

4.1 Manifest file

El manifest file es el archivo encargado del manejo del plugin y lo podemos encontrar en nuestro proyecto como “plugin.xml”.

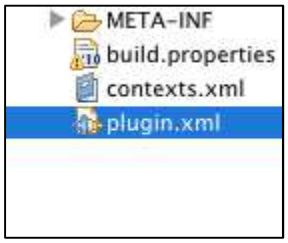


Figura 10. Estructura Eclipse - Plugin

Para comenzar a construir un plugin se debe tener claro que es lo que se pretende con él. En nuestro caso, el plugin debe incluir una nueva página de preferencias para controlar las propiedades del mismo, debe incluir un control de bucles for para detectar cuales pueden ser refactorizados y por último, debe incluir la funcionalidad para transformar los bucles for que desee el usuario.
Todo esto se consigue incluyendo en el archivo “plugin.xml” las extensiones que se deseen mediante la etiqueta “extensión point”.

```
<extension point="org.eclipse.ui.preferencePages">
	<page
		name="Bucles For Plug-in"
		class="plugin_preferences.PreferencePage"
		id="plugin_preferences.PreferencePage">
	</page>
</extension>
```

Ilustración 3. Extensión Preference Page

Como podemos observar, esta extensión permite incluir un submenú en el menú de preferencias de Eclipse que se nombra según lo que indique el campo "name" y que tratará toda su funcionalidad la clase indicada en el campo "class". Además, se ha incluido un identificador para localizar, si fuese necesario, esta extensión.

Para finalizar la página de preferencias de nuestro plugin se ha creado otra extensión para inicializar las variables. Esta funcionalidad es necesaria ya que en todas las páginas de preferencias existe un botón que restablece los valores iniciales y sin esta extensión no sería posible.

Para poder refactorizar un bucle for y transformarlo en la función paralel_for, es necesario incluir dos extensiones. La primera de ellas va a llamar a la clase encargada de encontrar todos los bucles for y mostrar únicamente aquellos que puedan ser refactorizados. Esto se ha conseguido de la siguiente forma:

```
<extension point="org.eclipse.cdt.codan.core.checkers">
	<checker
		class="plugin_proyecto.CrearMarcador"
		id="plugin_proyecto.CrearMarcador"
		name="For Statement">
		<problem
			category="org.eclipse.cdt.codan.core.categories.Security"
			defaultEnabled="true"
			defaultSeverity="Warning"
			description="Busqueda de bucles for"
			id="plugin_proyecto.forStatement"
				messagePattern="Puedes hacer un mejor uso de la
				función &amp;apos;&amp;apos;{0}&amp;apos;&amp;apos;"
			name="Uso de bucles for">
		</problem>
	</checker>
</extension>
```

Ilustración 4. Extensión Crear Marcador

Se ha creado una extensión en la que se llama a la clase "CrearMarcador" que es la encargada de analizar el código, obtener todos los bucles for y mostrar un marcador mediante el cual se indica la posibilidad de transformar dicho bucle. La etiqueta <problem> nos sirve para indicar el nivel de gravedad del problema encontrado y mostrar al usuario un mensaje indicando porque se ha marcado dicho bucle for.

En nuestro caso, hemos establecido que se avise al usuario la posibilidad de refactorizar un determinado bucle for mediante un "Warning". Además, se le mostrará un mensaje indicando la posibilidad de hacer un mejor uso de la función sobre la que se haya pinchado el marcador o se haya pasado el ratón por encima.

Para terminar, una vez mostrado al usuario la posibilidad de transformar una serie de bucles for que cumplan unas determinadas condiciones, es necesario llevar a cabo la refactorización. Para ello se ha incluido la siguiente extensión:

```
<extension point="org.eclipse.cdt.codan.ui.codanMarkerResolution">
        <resolution
                class="plugin_proyecto.TransformarCodigo"
                problemId="plugin_proyecto.forStatement">
        </resolution>
</extension>
```

Ilustración 5. Extensión Transformar Código

Con esta extensión, siempre que se pinche sobre un marcador y se acepte la transformación, se lleva a cabo la resolución del problema correspondiente. Esta resolución se trata en la clase TransformarCodigo.

Todas estas funcionalidades serán explicadas en los siguientes puntos detalladamente.

4.2 Página de preferencias

Como hemos comentado con anterioridad, para crear un nuevo submenú de preferencias es necesario la creación de dos extensiones en el manifest file. A su vez, se han creado tres clases para dicho submenú:

- ✓ PreferenceInitializer.java: se incluyen los valores iniciales de todos los elementos incluidos en el menú de preferencias.
- ✓ PreferencesConstants.java: se incluyen todos las constantes que se van a utilizar en el menú de preferencias.
- ✓ PreferencePage.java: es la clase encargada de la funcionalidad del nuevo submenú de preferencias.

Con el nuevo menú de preferencias se pretende modificar las propiedades del plugin, concretamente modificar el número de iteraciones a partir de los cuales los bucles for podrán ser refactorizados y la posibilidad de activar o desactivar el plugin cuando el usuario lo desee.

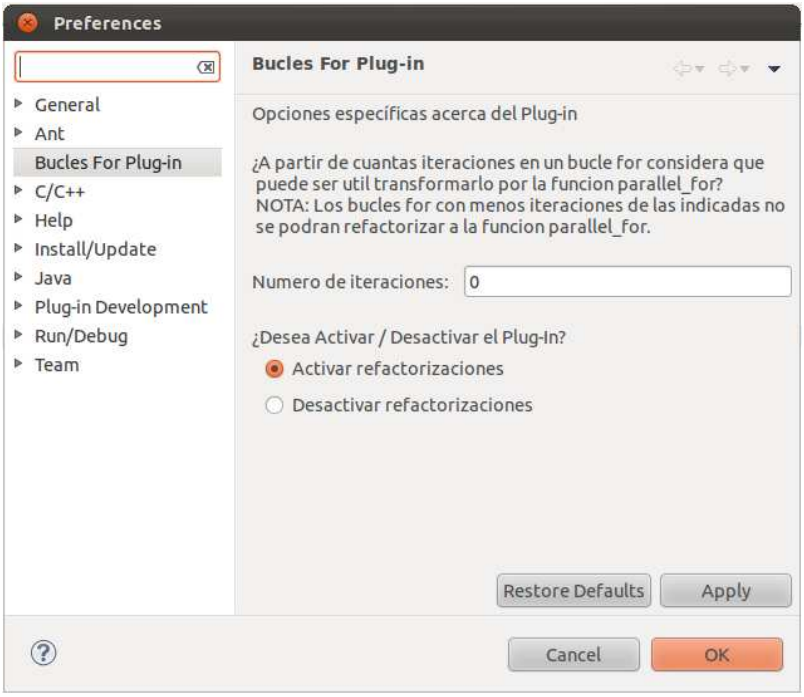


Figura 11. Menú de preferencias del plugin

Por tanto, la clase PreferencePage.java muestra un campo de texto para introducir el número de iteraciones y un radio button para activar o desactivar el plugin gracias al método createFieldEditors():

```
public void createFieldEditors() {
        label = new LabelFieldEditor("¿A partir de cuantas iteraciones en un
        bucle for considera que \n"  + …… , getFieldEditorParent());

        addField(label);

        iterationsEditor=new StringFieldEditor( PreferenceConstants.P_ITER,
        "&Numero de iteraciones: ", getFieldEditorParent());

        addField(iterationsEditor);

        activar_desactivar=new RadioGroupFieldEditor(
        PreferenceConstants.P_REFAC,"\n¿Desea Activar / Desactivar el Plug-
        In?", 1, …… , getFieldEditorParent());

        addField(activar_desactivar);
}
```

Ilustración 6. Código página de preferencias (I)

Además, también se encarga de controlar que el campo de texto del número de iteraciones solo contenga caracteres numéricos. En este sentido, si se introduce cualquier texto o se deja en blanco dicho campo Eclipse mostrará un mensaje de error y no permitirá al usuario guardar dichas propiedades deshabilitando los botones de "Apply" y "Ok".

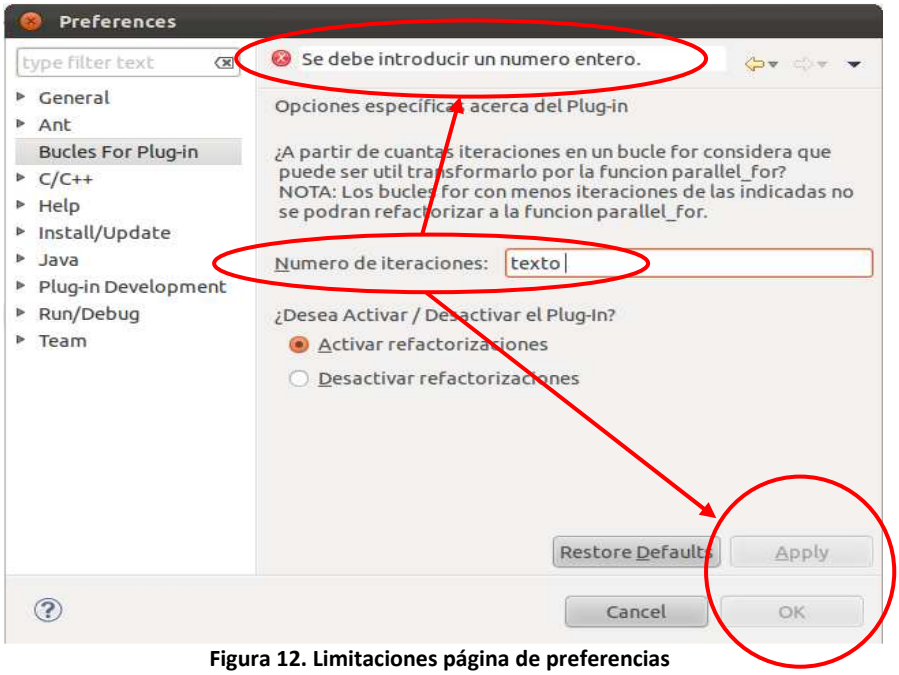


Figura 12. Limitaciones página de preferencias

Para controlar el contenido del campo de texto se ha utilizado la función checkState que extiende de la clase FieldEditorPreferencePage. Si no se introduce un valor numérico, se muestra al usuario un mensaje mediante el método setErrorMessage.

```
@Override
protected void checkState() {
        super.checkState();

        if(iterationsEditor.getStringValue()!= null &&
        !iterationsEditor.getStringValue().equals("")){
                try{
                        Integer.parseInt(iterationsEditor.getStringValue());
                        setErrorMessage(null);
                        setValid(true);

                }catch(NumberFormatException nfe){
                        setErrorMessage("Se debe introducir un numero
                        entero.");
                        setValid(false);
                }
        }
}
```

Ilustración 7. Código página de preferencias (II)

Así mismo, si se introduce los valores adecuados el usuario podrá pulsar el botón "OK" y los valores se guardarán en memoria. Dichos valores son transmitidos a la clase "CrearMarcador.java" para modificar los bucles for que pueden ser refactorizados, es decir, si se desactiva el plugin, la clase CrearMarcador.java lo recibe y no marca ningún plugin para dar al usuario la opción de refactorizar. Este tema lo trataremos con más detalle en el siguiente punto.

Los valores almacenados por el usuario los mantiene Eclipse en memoria para próximas sesiones, no teniendo que volver a modificar las preferencias del plugin cuando se vuelva a iniciar Eclipse.

Por último, el botón "Restore Defaults" modifica los valores del campo de iteraciones y el radio button de activar o desactivar a sus valores iniciales. Para ello, se ha creado la clase PreferenceInitializer.java en el que se incluyen dichos valores iniciales.

```
IPreferenceStore store = Activator.getDefault().getPreferenceStore();

store.setDefault(PreferenceConstants.P_ITER, "0");
store.setDefault(PreferenceConstants.P_REFAC, "activar");
```

Ilustración 8. Código página de preferencias (III)

4.3 Crear marcador

Para poder refactorizar los bucles for y transformarlos en la función parallel_for es necesario en primer lugar analizar el código y localizar todos los bucles for. Esta función la realiza la clase CrearMarcador.java. Para ello, se ha creado un patrón visitor que reconoce todas las expresiones del código y únicamente se queda con los bucles for.

```
public int visit(IASTStatement statements) {
        …
        if(statements instanceof IASTForStatement){

                IASTForStatement forState=(IASTForStatement)statements;
                …
}
```

Ilustración 9. Código crear marcador (I)

Una vez obtenidos todos los bucles for del código, se procede a establecer una serie de limitaciones impuestas por el cliente, como por ejemplo no marcar los bucles for incluidos en otros bucles for o incluidos en una función parallel_for. Otro tipo de limitaciones son las impuestas por la propia función parallel_for, como por ejemplo

que solo se podrán refactorizar bucles for con índices enteros y con incremento. En este sentido, para indicar que no se tiene que tratar dichos bucles for se utiliza la sentencia "PROCESS_CONTINUE". Por el contrario, si el bucle for se puede refactorizar, se utiliza la sentencia "PROCESS_SKIP" y el método "reportProblem" para indicar al usuario el problema por el que se ha marcado el bucle for.

Otra de las limitaciones son las impuestas por el usuario en el menú de preferencias. En este caso, la clase CrearMarcador.java obtiene el número de iteraciones del menú de preferencias y el valor del radio button activar/desactivar.

```
/* Se obtiene el valor de refactorizacion almacenado en memoria: activar / desactivar  */
activadoRefact=PreferencePage.getOpcionRefactor();

/* Se obtiene el numero de iteraciones almacenado en memoria (a partir de las cuales los bucles se pueden refactorizar)  */
iteracionesPosibles=Integer.parseInt(PreferencePage.getIterationsEditor());
```

Ilustración 10. Código crear marcador (II)

En este sentido, si el radio button "desactivar" se encuentra activado no se va a analizar el código para buscar los bucles for. Además, se va a comparar el número de iteraciones obtenidas de las preferencias con el número de iteraciones de cada bucle for del código y si este último es mayor se permite la refactorización.

Todos los bucles for que pueden ser refactorizados deben de ser reconocidos fácilmente por el usuario. Para ello, se procede a incluir un marcador por cada bucle for que pueda ser refactorizado y a subrayar dicho bucle.

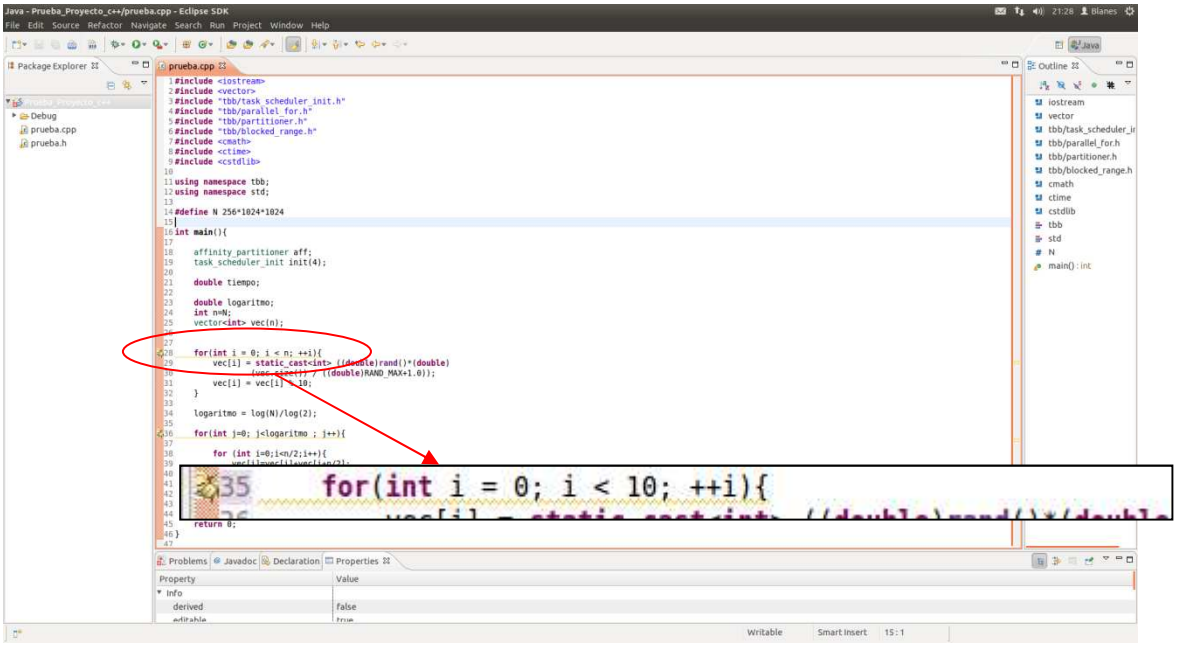


Figura 13. Marcador + subrayado

Como se puede observar el marcador indica un nivel medio de gravedad, es decir, se avisa al usuario la posibilidad de refactorizar mediante una marca de warning. Una vez se han reconocido los bucles que pueden ser transformados, el usuario tiene dos opciones:

- ✓ Pasar el ratón por encima del subrayado.
- ✓ Hacer clic sobre la marca de warning.

Si el usuario pasa el ratón por encima del subrayado se mostrará una descripción del problema. El mensaje que podrá ver el usuario es el siguiente:

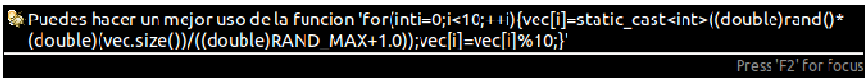


Figura 14. Descripción al pasar el ratón por encima

Por el contrario, si el usuario hace clic sobre el marcador se mostrará el siguiente cuadro de diálogo:

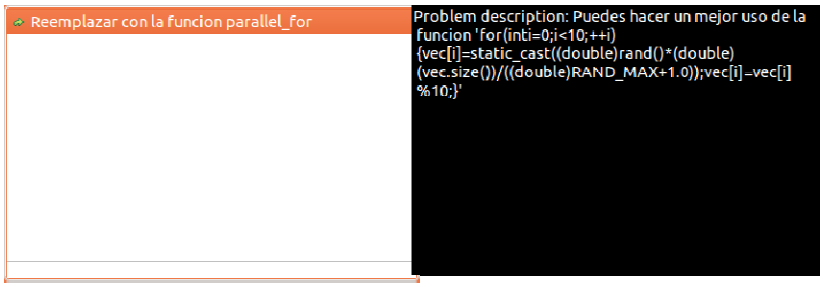


Figura 15. Cuadro de diálogo al hacer clic sobre el marcador

La primera opción sólo muestra al usuario una simple información del problema, pero la segunda a parte de la descripción ofrece al usuario la posibilidad de llevar a cabo la transformación. Si el usuario desea llevar a cabo la refactorización debe de hacer clic sobre la frase "Reemplazar con la función parallel_for", transformando el bucle for en la función parallel_for de la biblioteca Intel TBB como se explicará en el punto 4.4.

4.4 Transformar código

Una vez el usuario hace clic sobre la frase "Reemplazar con la función parallel_for" del cuadro de diálogo, el plugin transforma el bucle for a la función parallel_for gracias a la clase transformarCodigo.java.

Esta clase se encarga de obtener los datos necesarios del bucle for, es decir, se encarga de obtener el rango, el incremento, la condición, el contenido, el nombre de la variable…

```
for (int i=0;i< 10;++i){
        vec [ i ] = 0;
}
```

Ilustración 11. Partes del bucle for

Además, también debe obtener el árbol sintáctico, el nodo y el marcador correspondiente al bucle for que se desea refactorizar.

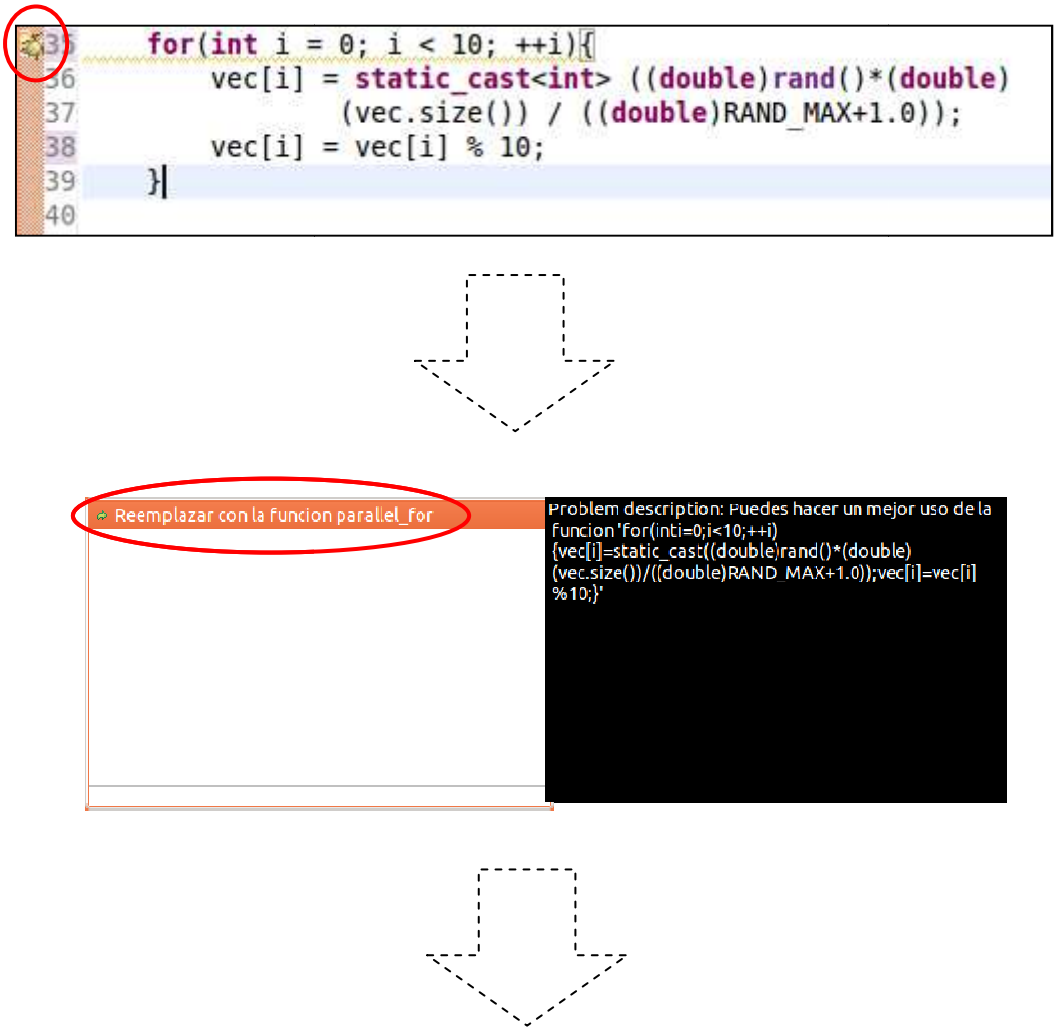
Una vez obtenido los datos necesarios, se modifica el árbol sintáctico abstracto (AST) mediante el método reescribirAST que utiliza los datos obtenidos anteriormente.

```
public void reescribirAST(String nuevoString,IASTTranslationUnit
ast,IASTNode nodo, IMarker marker){
        ASTRewrite r = ASTRewrite.create(ast);
        INodeFactory factory = ast.getASTNodeFactory();

        IASTIdExpression id = factory.newIdExpression
        (factory.newName(nuevoString.toCharArray()));

        IASTStatement newStatement = factory.newExpressionStatement(id);

        r.replace(nodo, newStatement, null);

        Change c = r.rewriteAST();
        try {
                c.perform(new NullProgressMonitor());
        } catch (CoreException e) {
                return;
        }
        try {
                marker.delete();
        } catch (CoreException e) {
                return;
        }
}
```

Ilustración 12. Código transformar código

Gracias al método reescribirAST, el bucle for seleccionado se transforma en la función parallel_for. Este método recibe el nuevo String (función parallel_for) ya formado previamente y por tanto, elimina el bucle seleccionado a través del marcador y enlaza el árbol sintáctico a esta nueva función.

Como podemos ver en las siguientes imágenes, un simple bucle for se transforma en la función parallel_for de manera rápida, sencilla y transparente al usuario aunque existen bastantes pasos previos para llevarlo a cabo.




```
34    // TODO Auto-generated method
35    parallel_for(blocked_range<int>(0, 10), [&](const blocked_range<int> &r) {
36        for(int i = r.begin();i < r.end(); ++i) {
37            vec[i] = static_cast<int> ((double)rand()*(double)
38                    (vec.size()) / ((double)RAND_MAX+1.0));
39            vec[i] = vec[i] % 10;
40        }}, aff);
41
```

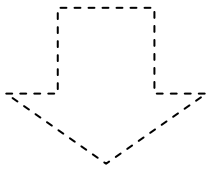
La nueva función parallel_for utiliza los datos obtenidos previamente (rango, nombre de variables,…), se elimina el marcador que tenía asociado y se inserta un comentario para indicar al usuario que se trata de una función autogenerada.

Además, la clase TransformarCodigo.java tras llevar a cabo la transformación realiza un análisis del código para comprobar si existen todas las sentencias necesarias para compilar y ejecutar esta nueva función.

Estas sentencias son los includes, el namespace, el task_scheduler_init y el affinity_particioner.

Como hemos comentado, se lleva a cabo el análisis del código y si cualquiera de estas sentencias no existen se crearían automáticamente. Este proceso se puede ver a continuación:

```
 7
 8 #include <iostream>
 9 #include <vector>
10 #include "tbb/task_scheduler_init.h"
11 #include "tbb/blocked_range.h"
12 #include <cmath>
13 #include <ctime>
14 #include <cstdlib>
15
16 using namespace tbb;
17 using namespace std;
18
19 #define N 256*1024*1024
20
21 int main(){
22
23     affinity_partitioner aff;
24     task_scheduler_init init(4);
25
```



```
 7
 8 #include <iostream>
 9 #include <vector>
10 #include "tbb/task_scheduler_init.h"
11 #include "tbb/blocked_range.h"
12 #include <cmath>
13 #include <ctime>
14 #include <cstdlib>
15 #include "tbb/parallel_for.h"
16 #include "tbb/partitioner.h"
17
18 using namespace tbb;
19 using namespace std;
20
21 #define N 256*1024*1024
22
23 int main(){
24
25     affinity_partitioner aff;
26     task_scheduler_init init(4);
27
```

Figura 17. Proceso de autocompletar código

Como podemos ver, al realizar la transformación no existian los includes necesarios para soportar la función parallel_for y el affinity_particioner, pero la clase TransformarCodigo analiza el código y lo detecta.

El sistema, al encontrar la última sentencia "#include", introduce aquellas no encontradas en las siguientes líneas. De esta froma, gracias a la función autocompletar código el usuario no se tiene que preocupar de buscar e introducir las sentencias necesarias para ejecutar la función parallel_for.

Para terminar, la función parallel_for necesita que los valores del bucle for que se va a paralelizar sean valores enteros ya que dicha función unicamente funciona con esos valores. En este sentido, nuestro plugin va a reconocer varios tipos de bucles for:

- ✓ Aquellos que tienen indices enteros.
- ✓ Aquellos que contienen un iterador de un vector de número enteros.

Los bucles for que tienen índices enteros son los más básicos y son los que hemos tratado con anterioridad. Pero el otro caso todavía no lo hemos tratado en esta memoria.
Nuestrol plugin es capaz de transformar los bucles for que contienen un iterador que recorre un vector de números enteros. Para ello, el plugin se encarga de analizar el código, buscar el vector sobre el que hace referencia el iterador y obtener su rango. Es decir, nuestro plugin es capaz de reconocer bucles for como el siguiente:

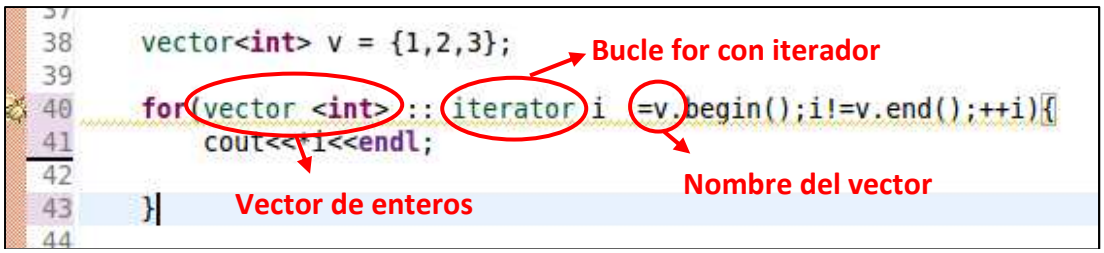


Figura 18. Partes del bucle for con iterador

El plugin reconoce que se trata de un bucle for con un iterador sobre un vector de enteros de nombre "v".

Entonces, se busca en el código la estructura "vector<int>" + nombre del vector + "=" + "{" + valores del vector + "}" + ";". Una vez encontrada, se lleva a cabo la refactorización con los valores del vector de enteros.

```
37
38    vector<int> v = {1,2,3};
39
40    // TODO Auto-generated method
41    parallel_for(blocked_range<int>(1, 3), [&](const blocked_range<int> &r) {
42        for(int i = r.begin();i != r.end();++i) {
43            cout<<*i<<endl;
44
45        }}, aff);
46
```

Figura 19. Parallel_for (Bucle for con iterador)

Solo se reconocen bucles for con iterador sobre vector de enteros declarados de la forma explicada anteriormente. Si existe un iterador sobre una lista, por ejemplo, nuestro plugin no permitirá la refactorización. Estas limitaciones se han considerado trabajos futuros como se ecplicará en el capítulo 6.

Además, nuestro plugin permite refactorizar bucles for que contienen variables enteras, es decir, existen determinados casos que por su complejidad no se han comprobado su veracidad y dejamos al usuario la responsabilidad de un correcto uso.
Esto es debido a que se pude comprobar que rango esta introduciendo el usuario, pero no podemos saber si introduce una variable entera correctamente declarada, un double no permitido en la función parallel_for,... Por lo tanto, siempre que no sea un valor entero se va a proceder a aceptar la refactorización siempre y cuando no se lo impida otra limitación.

Capítulo 5

Manual de instalación y de usuario

5.1 Manual de instalación

En la presente sección se proporcionan las instrucciones necesarias para instalar el complemento de refactorización en Eclipse.

5.1.1 Requisitos de arquitectura

El complemento de refactorización para Eclipse que estamos creando sirve para paralelizar código C++ secuencial utilizando la biblioteca Intel TBB.

Por lo tanto, para poder instalar y utilizar el plugin, es necesario disponer de los siguientes elementos instalados previamente:

- ✓ Eclipse CDT (entorno de desarrollo de programas en C/C++).
- ✓ Biblioteca Intel TBB.

5.1.2 Instalación Eclipse CDT

Para instalar el entorno de desarrollo es necesario seguir las siguientes instrucciones:

1. Acceder desde un navegador a la siguiente web:

   http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers-includes-incubating-components/indigosr2

2. Elegir en la parte derecha de la web el Sistema Operativo deseado.

3. Descargar Eclipse.

4. Descomprimir el fichero descargado.

5. En la carpeta descomprimida "eclipse", existe un fichero ejecutable "eclipse" para iniciar el programa.

6. Acceder a las propiedades del proyecto dentro de Eclipse y en el submenú C/C++ Build/Settings en las opciones del compilador de GCC "Gcc C++ Compiler" se tiene que incluir la sentencia -std=c++0x y la librería de tbb. Para poder insertarlos, se tiene que escribir en Miscellaneous las siguientes sentencias:

   ```
   -l/home/Usuario/Desktop /tbb40_20120201oss/include
   -std=c++0x
   ```

5.1.3 Instalación Intel TBB

Así mismo, para instalar la biblioteca Intel TBB es necesario seguir las siguientes instrucciones:

1. Acceder a la página web de Intel TBB y descargar la última versión estable de la pestaña “Stable release” (TBB 4.0):

   http://threadingbuildingblocks.org/download.php

2. Descomprimir el fichero descargado.

3. Ingresar en la carpeta que acabamos de descomprimir .

4. Ejecutar el comando “gmake” para compilar las librerías.

5.1.4 Instalación del complemento de refactorización

Una vez terminada la instalación de los elementos anteriores, procederemos a la instalación del plugin por medio de archivos, es decir, al instalar el Eclipse se generan una serie de carpetas en las que hay que incluir el archivo del plugin.

Siga los siguientes pasos para realizar la instalación del programa:

1. Copie el archivo "Plugin_Proyecto_1.0.0.201208251147" localizado en la carpeta "Desarrollo_plugin/producto_final/plugins/".

2. Mover dicho archivo a la carpeta “plugins” del directorio donde se ha instalado el Eclipse CDT (home/usuario/Desktop/eclipse/plugins/).

3. Para terminar, si tienes ejecutándose eclipse, deberás reiniciarlo para que surtan efecto los cambios.

4. Comienza a utilizar el plugin de refactorización de código C++ secuencial a C++ paralelo.

5.2 Manual de usuario

En esta sección se incluyen todas las instrucciones para poder utilizar el complemento de refactorización correctamente.

Para comenzar a utilizar el complemento de refactorización es necesario conocer como es el entorno de desarrollo Eclipse y que aspectos de él nos son interesantes para utilizar el plugin.

Como podemos ver en la siguiente imagen, existen tres zonas de alto interés para el usuario y que a continuación explicaremos cómo usarlas.

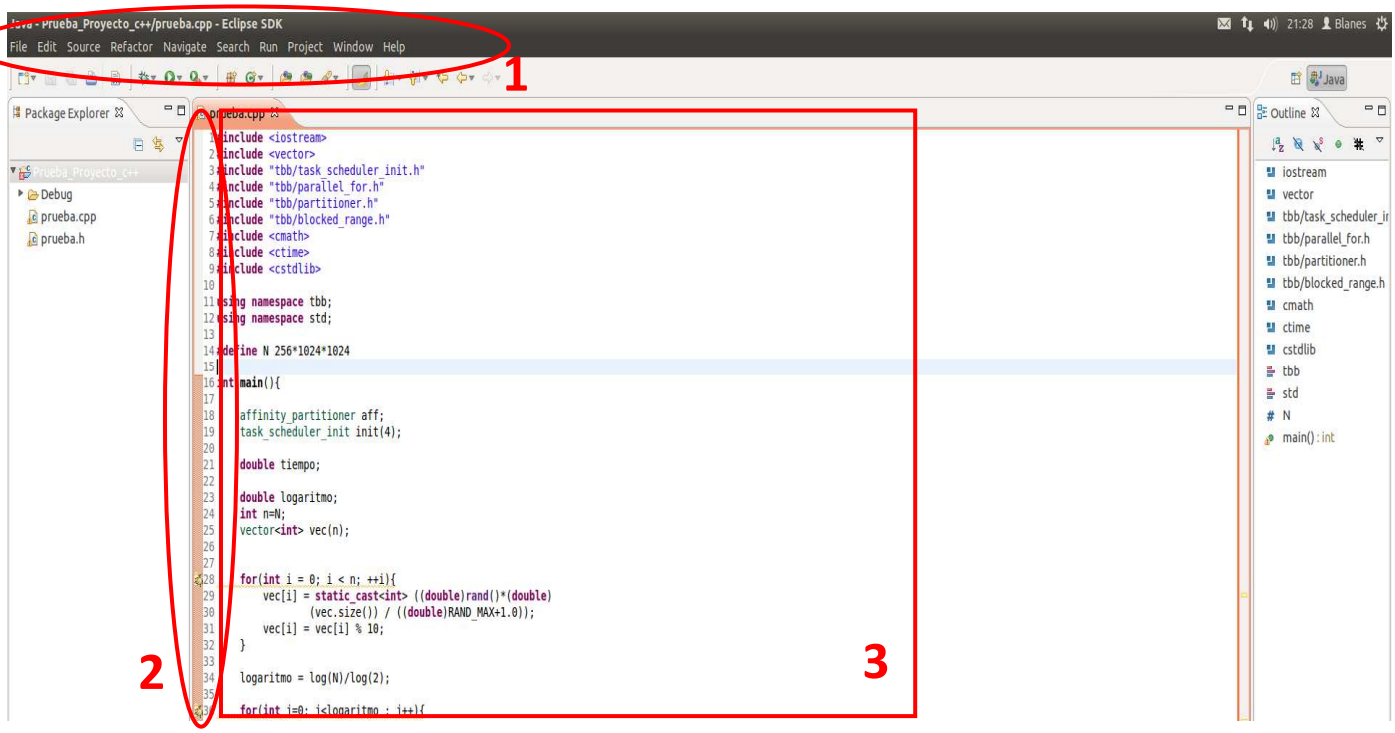
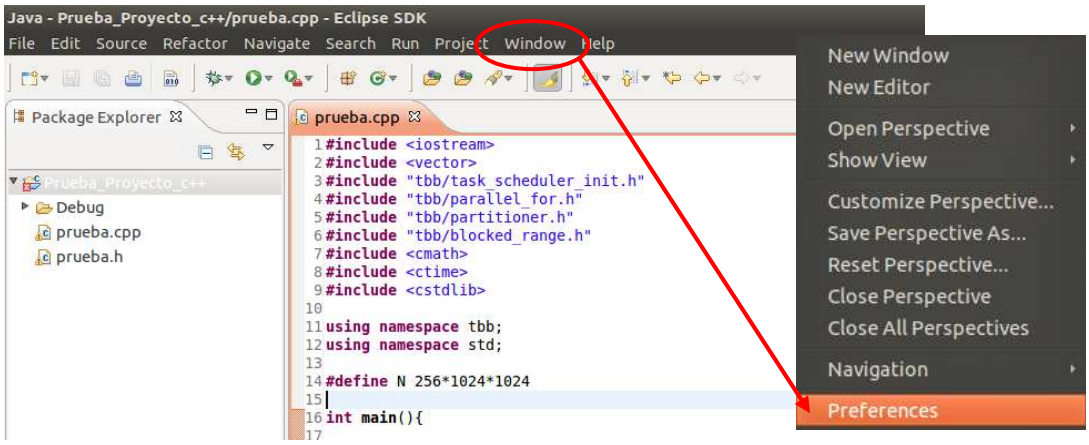


Figura 20. Manual de usuario. Pantalla inicio Eclipse

La primera zona es dónde se puede acceder a las preferencias de Eclipse y de esta forma modificar las propiedades del plugin. Mientras, la segunda zona es donde se muestra al usuario la marca de que un plugin es refactorizable y dónde él mismo puede hacer clic para realizar la transformación. Por último, la tercera es el editor del código de eclipse. En él podemos observar cómo se subrayan los bucles que se pueden transformar, así como la transformación propiamente dicha.

El menú de preferencias del plugin se podrá acceder a través del menú “Window” ubicado en la “primera zona” del entorno de desarrollo Eclipse, seleccionando la opción “Preferences” de dicho menú.



Una vez en el menú de preferencias, debes seleccionar el submenú “Bucles For Plug-In”.

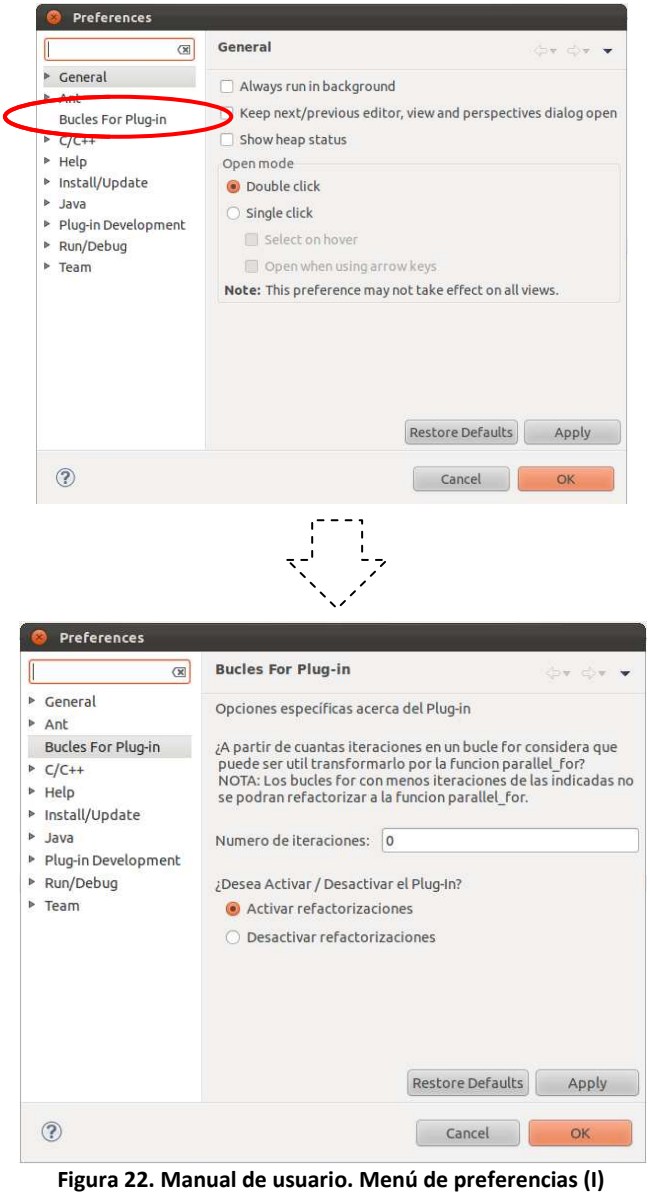


Figura 22. Manual de usuario. Menú de preferencias (I)

Como podemos ver, el menú de propiedades del plugin está compuesto por dos opciones, modificar el número de iteraciones a partir de las cuáles los bucles for podrán refactorizarse y activar o desactivar el funcionamiento del plugin.

El número de iteraciones debe ser un carácter numérico, no pudiendo introducir ningún tipo de texto ni dejar el campo en blanco. En este sentido, si el usuario introduce cualquier carácter que no sea numérico o deja el campo de texto en blanco el plugin lo detectará y mostrará un mensaje de error. Además, tampoco dejará aceptar las propiedades inhabilitando los campos de OK y Apply.

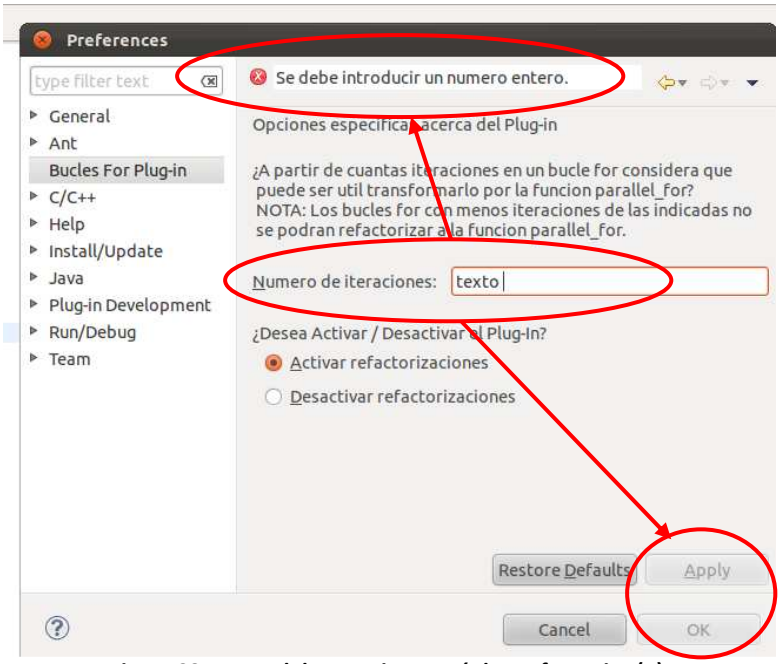


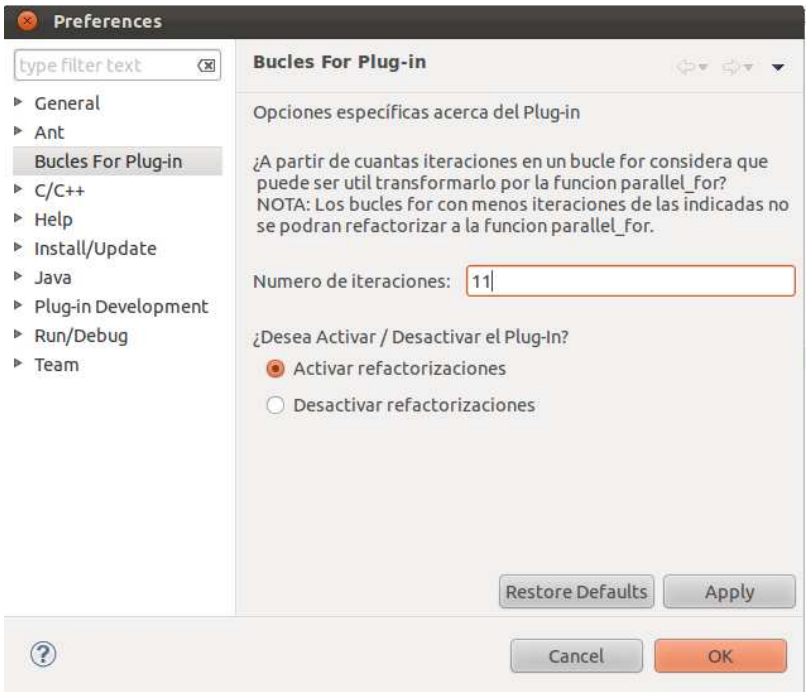
Figura 23. Manual de usuario. Menú de preferencias (II)

En este sentido, si tenemos un bucle for con 10 iteraciones en un primer instante se podrá transformar en la función parallel_for ya que el número de iteraciones inicial es 0 para que se puedan refactorizar todos los bucles for independientemente del número de iteraciones. Pero si tu deseas que únicamente se refactoricen aquellos bucles for con más de 11 iteraciones, el bucle anterior no dará la opción al usuario de ser refactorizado. Es decir, si disponemos de un bucle for como el siguiente:

```
27
28      for(int i = 0; i < 10; ++i){
29          vec[i] = static_cast<int> ((double)rand()*(double)
30                  (vec.size()) / ((double)RAND_MAX+1.0));
31          vec[i] = vec[i] % 10;
32      }
```

Figura 24. Manual de usuario. Señalización bucles for

Y modificamos las propiedades del plugin de tal forma que en el campo de texto del número de iteraciones introduzcamos un 11, el bucle for anterior no se mostrará al usuario como refactorizable:




```
for(int i = 0; i < 10; ++i){
    vec[i] = static_cast<int> ((double)rand()*(double)
            (vec.size()) / ((double)RAND_MAX+1.0));
    vec[i] = vec[i] % 10;
}
```

Figura 25. Manual de usuario. Cambio número de iteraciones

Por otro lado, el usuario podrá activar o desactivar el plugin utilizando el radio button. Esta funcionalidad es muy importante si el usuario no está interesado en paralelizar el código ya que de esta forma no se le molestará con advertencias que no tendrá en cuenta al no estar interesado.

Si se activa el plugin, los bucles for correspondientes se muestran al usuario como refactorizables. Es decir, se muestra un marcador y se subrayan los bucles for correspondientes. Sin embargo, si el plugin se desactiva, ningún bucle for podrá ser refactorizado no señalando de ninguna forma los bucles for.

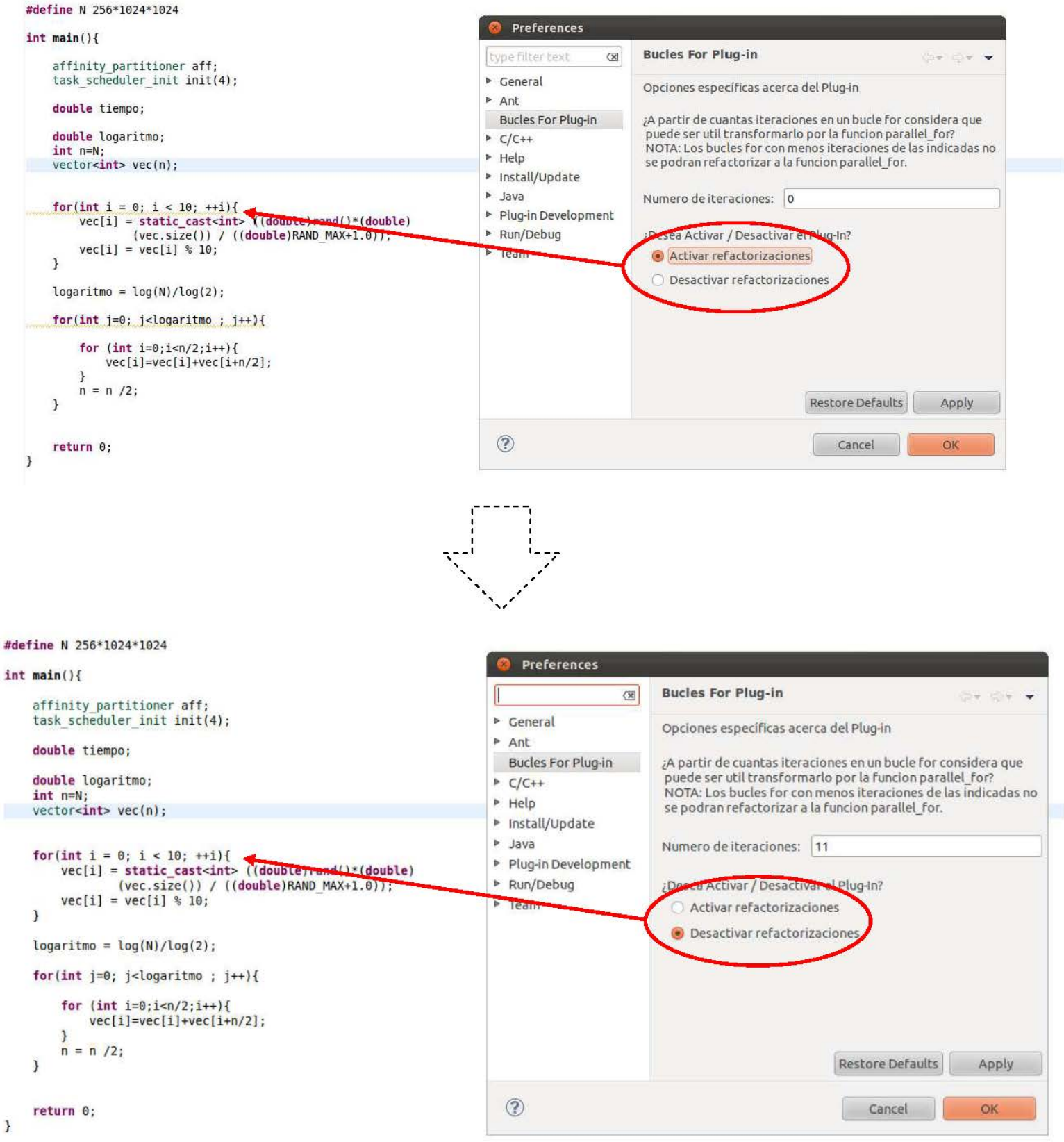


Figura 26. Manual de usuario. Cambio activar/desactivar

Para terminar con el menú de preferencias del plugin, existe el botón "Restore Defaults" que si el usuario lo pulsa devuelve el valor inicial de todas las propiedades. En nuestro caso, si se pulsa dicho botón se introducirá un 0 en el número de iteraciones y se activaran las refactorizaciones.

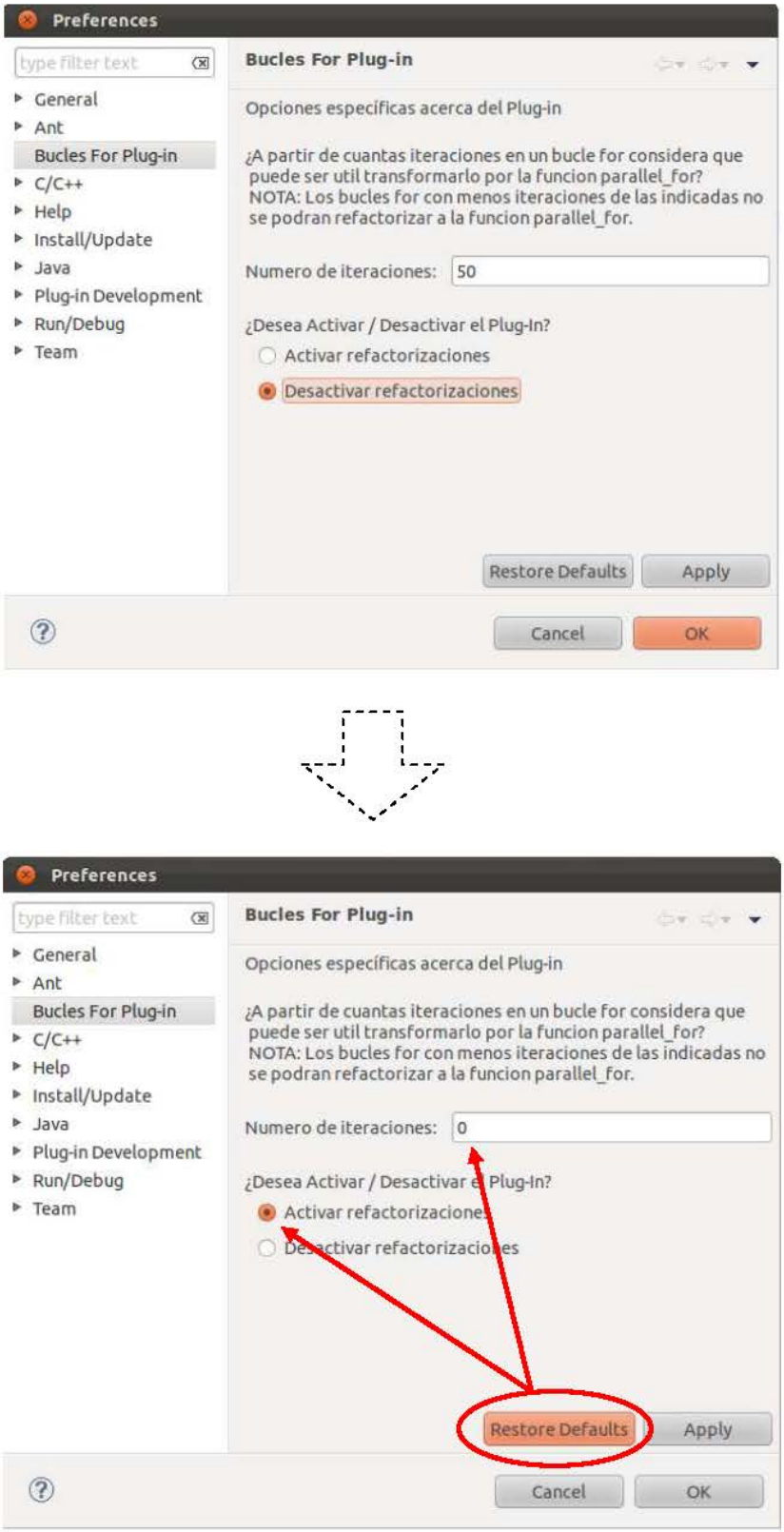


Figura 27. Manual de usuario. Valores por defecto

En la segunda zona marcada anteriormente en la pantalla inicial de Eclipse se marca al usuario la posibilidad de refactorizar. Esta marca es de gravedad media, es decir, se muestra como una marca de "Warning".

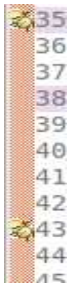


Figura 28. Manual de usuario. Marcadores

Además de marcar en la segunda zona, también se subraya el bucle for correspondiente en la tercera zona (editor de Eclipse).

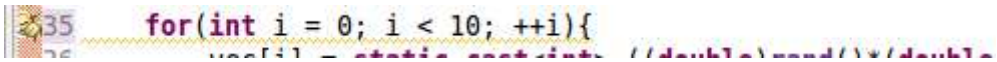


Figura 29. Manual de usuario. Subrayado

Si el usuario únicamente desea conocer la descripción del problema encontrado basta con pasar el ratón por encima del bucle for que se desee. En este caso, se muestra un mensaje al usuario con la descripción del problema encontrado:

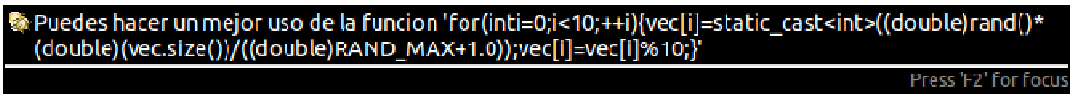


Figura 30. Manual de usuario. Descripción

Por el contrario, si desea conocer la descripción pero también tener la posibilidad de llevar a cabo la transformación se debe pulsar sobre el marcador.

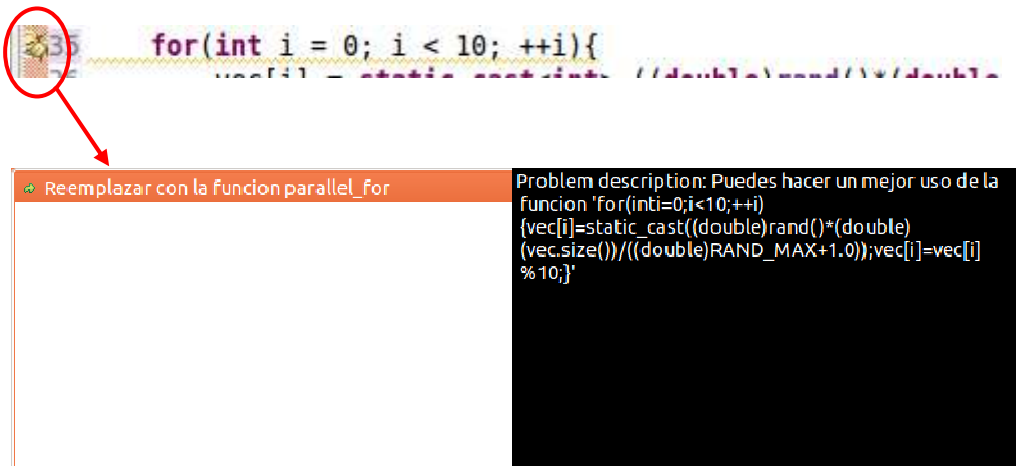


Figura 31. Manual de usuario. Opción de refactorización

Una vez pulsado sobre el marcador, se puede cancelar la transformación pinchando sobre cualquier punto fuera del cuadro de diálogo, o aceptar la transformación pinchando sobre la frase “Reemplazar con la función parallel_for”.

Si se pincha sobre dicha frase el bucle for correspondiente se convierte en la función parallel_for de la biblioteca Intel TBB, paralelizando el código de una forma transparente al usuario.

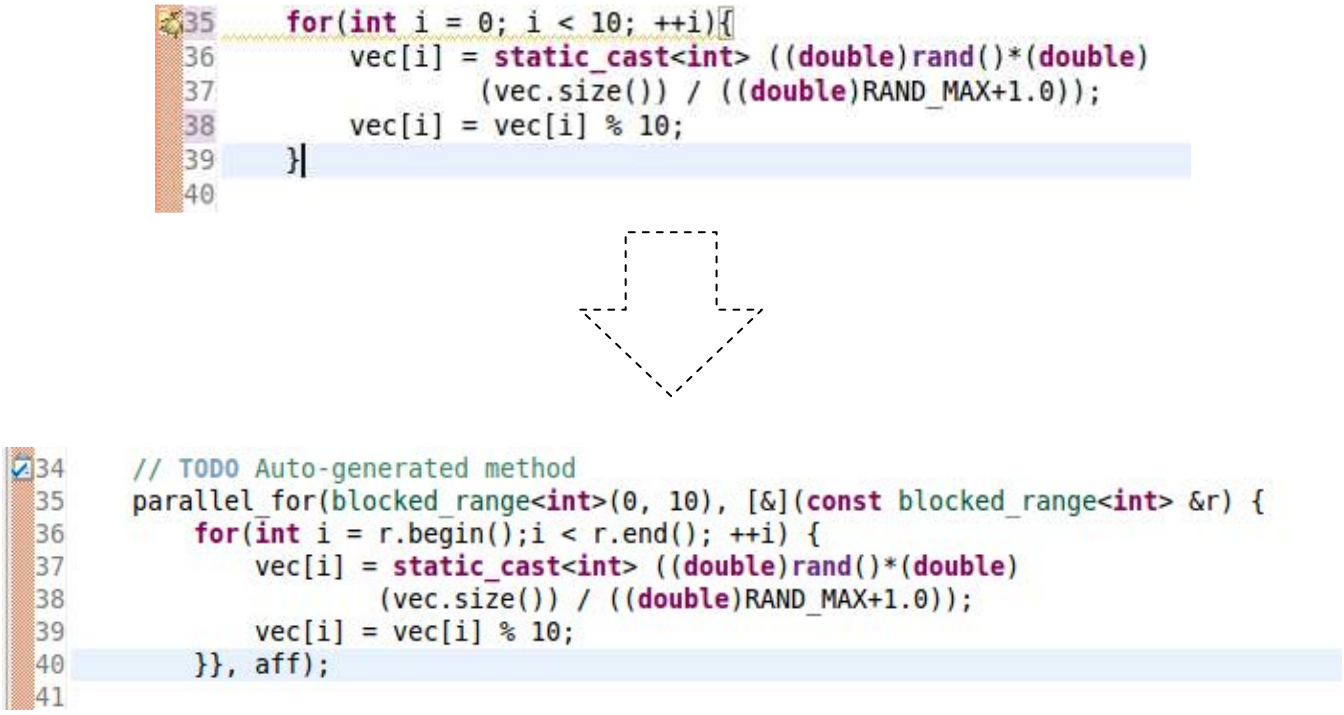


Figura 32. Manual de usuario. Proceso de refactorización

De esta forma el código ha sido paralelizado y además se le muestra un comentario al usuario indicándole que el código ha sido autogenerado: “// TODO Auto-generated method”.

Una vez se ha finalizado la transformación, el plugin inicia la etapa de autocompletar el código. Para ello, analiza el código y si observa que no se dispone de alguna sentencia necesaria para compilar y ejecutar la función parallel_for, se incluye en el código. Esta función también es transparente al usuario, no teniéndose que preocupar por ello.

Para terminar, el usuario también es capaz de paralelizar bucles for que contienen un iterador sobre un vector de enteros. Es decir, este tipo de bucles for también los reconoce el plugin como se ha explicado en el capítulo anterior.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1 Conclusiones

Como se ha explicado en la sección “Objetivos” del primer capítulo, el principal objetivo de este proyecto era mejorar el tiempo de ejecución de la aplicación a desarrollar sin aumentar el tiempo de programación gracias a la paralelización del código, concretamente de los bucles for. A lo largo del documento se ha podido seguir la correcta evolución del proyecto, observando cómo se ha conseguido este objetivo con la elaboración completa del complemento de refactorización.

Además, se han cumplido una serie de objetivos secundarios expuestos anteriormente e incluso otros que no se habían propuesto inicialmente:

- ✓ El sistema deberá mostrar al usuario cuando es posible la refactorización de una parte del código.
- ✓ El sistema mostrará una descripción de la refactorización antes de aceptar la misma.
- ✓ El sistema deberá autocompletar el código con las sentencias necesarias cuando se lleve a cabo la refactorización.
- ✓ El sistema deberá ofrecer al usuario desactivar el plugin por si este no desea utilizarlo.
- ✓ El sistema deberá ofrecer al usuario la posibilidad de modificar el número de iteraciones de los bucles for a partir de las cuales se permitirá la refactorización, despreciando el resto de bucles.
- ✓ El sistema no permitirá refactorizar bucles for con decremento.
- ✓ El sistema permitirá transformar bucles for con un iterador sobre un vector de números enteros
- ✓ El sistema permitirá transformar bucles for con una variable de tipo entero.
- ✓ El sistema mostrará al usuario mediante un comentario que el código refactorizado se ha autogenerado.

Como se puede observar, se han cumplido de forma adecuada todos los objetivos que se habían expuesto inicialmente incluso existe la posibilidad de continuar con el desarrollo de este proyecto con trabajos futuros que se verán en el siguiente punto.

6.2 Trabajos futuros

Gracias a la multitud de opciones que ofrece la biblioteca Intel TBB para paralelizar código c++ secuencial, existe la posibilidad de ampliar y mejorar este proyecto.

El principal objetivo de este complemento de refactorización es mejorar el tiempo de ejecución de cualquier aplicación mediante la paralelización de código c++ secuencial. En la mayoría de aplicaciones existentes, el tiempo de cómputo crece considerablemente en los bucles. Con este proyecto se pretende mejorar este problema mediante la paralelización de los bucles for que se encuentren en el código, pero existen otros bucles que también podrían ser refactorizados. Los otros bucles que podrían ser paralelizados serían:

- ✓ Bucles while.
- ✓ Bucles do-while.

Por tanto, la primera idea para mejorar el plugin es incluir la posibilidad de refactorizar bucles while y do-while mediante funciones incluidas en la biblioteca Intel TBB. Es decir, se trata de realizar transformaciones de bucles while mediante la función parallel_while y de bucles do-while mediante la función parallel_do.

Otro de los posibles trabajos futuros es una mejora en el analizador de código, es decir, se trataría de mejorar el análisis de código para permitir refactorizar un mayor número de bucles for. Esto es debido a que solo se puede introducir en el rango de la función parallel_for números enteros, por lo que se ha establecido una limitación a la hora de seleccionar los bucles for. Los bucles for con índices enteros son tratados de manera adecuada, pero si existe una variable entera o un iterador en el bucle se debe llevar a cabo un análisis de código y es aquí donde radica la posible mejora.

En el caso de los iteradores, solo se ha programado el plugin para reconocer iteradores sobre vectores de tipo entero. En este sentido, se puede mejorar el complemento de refactorización para que se reconozcan, por ejemplo, listas de números enteros.

Pero en el caso de las variables enteras existe una mejora fundamental. En este plugin solo se reconoce los rangos de los bucles for del tipo entero o cuando se utiliza un iterador, si esto no fuese así se procede a permitir la refactorización salvo que otra limitación lo impida. Esto es debido ya que es computacionalmente complejo reconocer si es una variable entera declarada correctamente o es una variable double no permitida en la función parallel_for. Se podría introducir un analizador de código más potente y reconocer todo estos distintos tipos.

6.3 Presupuesto

El coste del análisis, diseño y desarrollo del proyecto viene establecido por la duración del mismo y por otros gastos indirectos. Todos estos gastos se desglosarán a continuación, teniendo en cuenta los siguientes datos:

Datos de personal				
Duración del proyecto (días)	Horas diarias	Dedicación hombre mes	Coste hombre mes	Coste (Euro) [1]
98	4	131,25	2694,39	8047,24
			Total	***8047,24***
[1] Fórmula del cálculo del coste de personal: Coste= ((duración días * horas días) / dedicación hombre mes) * coste hombre mes				

Tabla 41. Presupuesto. Datos de personal

Además, para la realización del proyecto ha sido necesario adquirir los siguientes equipos con su correspondiente coste de amortización:

Equipos					
Descripción	Coste(Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de deprecación	Coste imputable [2]
Ordenador Intel Core i5-2400	480,00	100	5	60	24,00
				Total	***24,00***
[2] Fórmula de cálculo de la Amortización: $\frac{A}{B} \times C \times D$ **A =** nº de meses desde la fecha de facturación en que el equipo es utilizado **B =** periodo de depreciación (60 meses) **C =** coste del equipo (sin IVA) **D =** % del uso que se dedica al proyecto (habitualmente 100%)					

Tabla 42. Presupuesto. Equipos

Por otro lado se van a detallar otros gastos directos del proyecto que no han sido contemplados en los conceptos anteriores como pueden ser fungibles, viajes y dietas, otros,…

Otros gastos directos		
Descripción	Empresa	Costes imputables
Internet	Telefónica	200,00
Luz	Iberdrola	100,00
Microsoft Office 2010 Estudiantes	Microsoft	140,00
	Total	***440,00***

Tabla 43. Presupuesto. Otros gastos directos

Además, no se ha necesitado ninguna subcontratación para realizar alguna de las tareas del proyecto y hemos tenido en cuenta una tasa del 20% para calcular los gastos indirectos.

El coste total del proyecto, de forma resumida, lo podemos comprobar en la siguiente tabla:

Resumen de costes	
Descripción	Presupuesto costes totales
Gastos de personal	8047,24
Gastos de equipo (Amortización)	24,00
Subcontratación de tareas	0
Otros gastos directos	440,00
Gastos indirectos (20 %)	1702,25
Total Sin IVA	***10213,49***
Total Con IVA (18 %)	***12051,92***

Tabla 44. Presupuesto. Resumen de costes

El presupuesto total del proyecto asciende a la cuantía de **DOCE MIL CINCUENTA Y UN EUROS CON NOVENTA Y DOS CÉNTIMOS DE EURO**.

Anexo 1

Eclipse PDE

Eclipse PDE es el entorno de desarrollo de plugins de Eclipse. Para comenzar a utilizarlo es necesario descargarte la última versión de Eclipse disponible y descomprimirla en tu PC. Una vez descomprimida, acceder a la carpeta "eclipse" y ejecutar el programa.

En estos primeros pasos, vamos a utilizar el asistente que nos provee Eclipse para que nos cree el esqueleto de lo que va a ser nuestro primer plugin para Eclipse.

El primer paso es crear nuestro proyecto en Eclipse, para ello, seleccionamos "Nuevo proyecto" y en el wizard seleccionamos Plug-in Project.

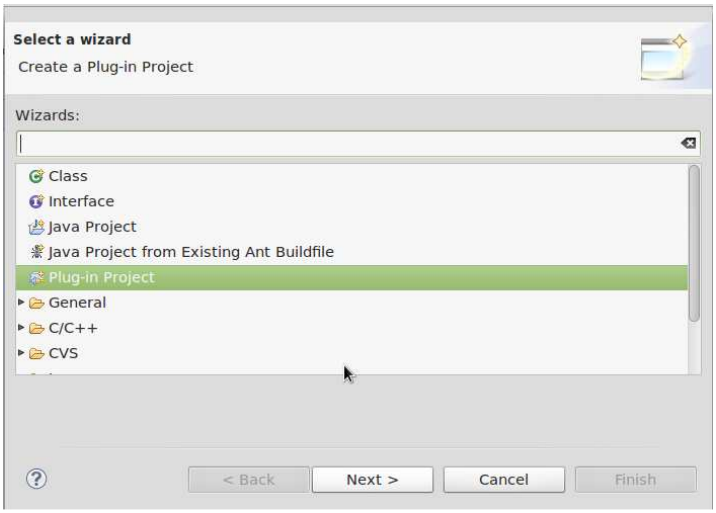


Figura 33. Eclipse PDE (I)

Pulsamos en el botón Next y en la siguiente pantalla debemos introducir el nombre de nuestro proyecto. En este caso vamos a crear el esqueleto para crear una nueva página de preferencias como en nuestro proyecto, por lo que el nombre del proyecto será "PreferencesPage". Dejamos el resto de opciones por defecto y pulsamos Next.

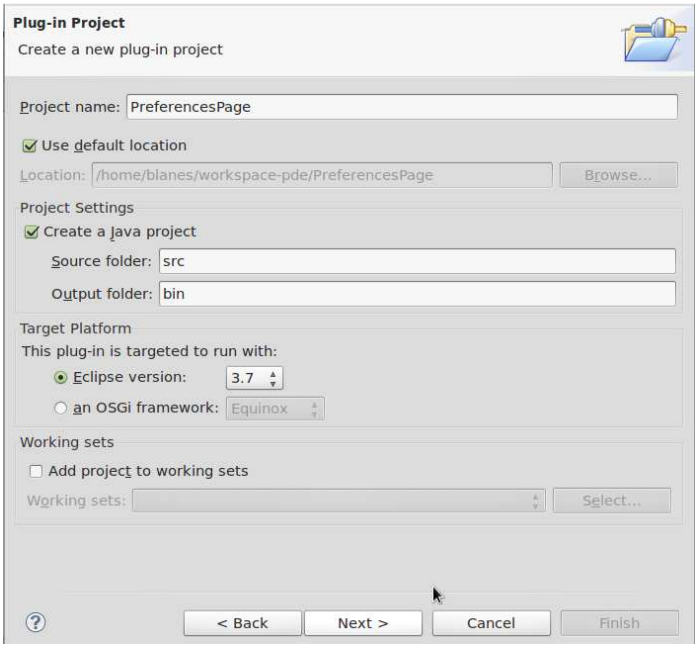


Figura 34. Eclipse PDE (II)

Introduces las propiedades del plugin, nombre, id, versión,... y pulsas Next.

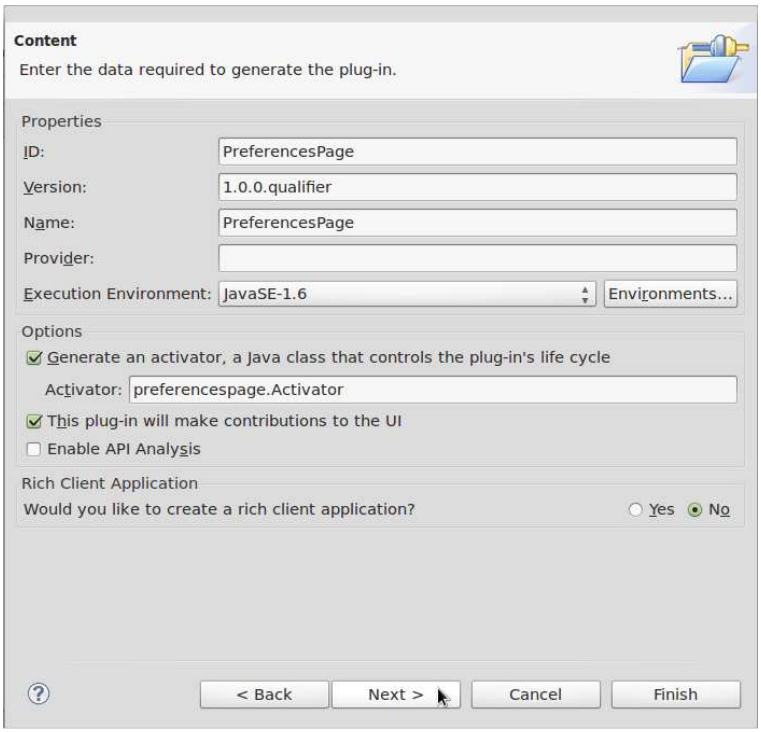


Figura 35. Eclipse PDE (III)

Y después, se debe elegir Custom plug-in wizard para personalizar nuestro plugin y de nuevo se pulsa Next.

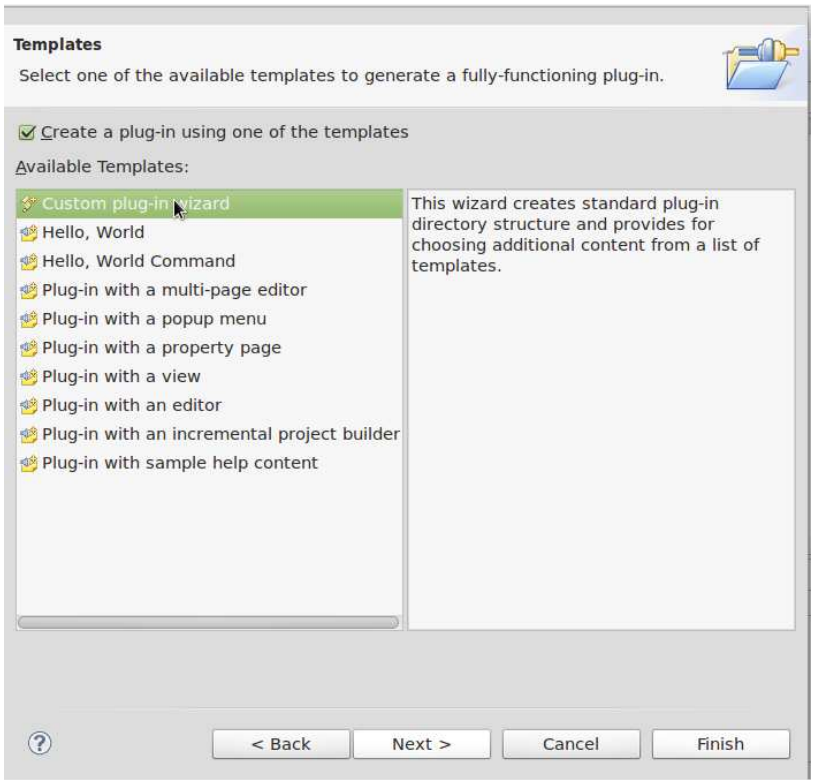


Figura 36. Eclipse PDE (IV)

Se elige la opción "Preference Page" y pulsamos Finish.

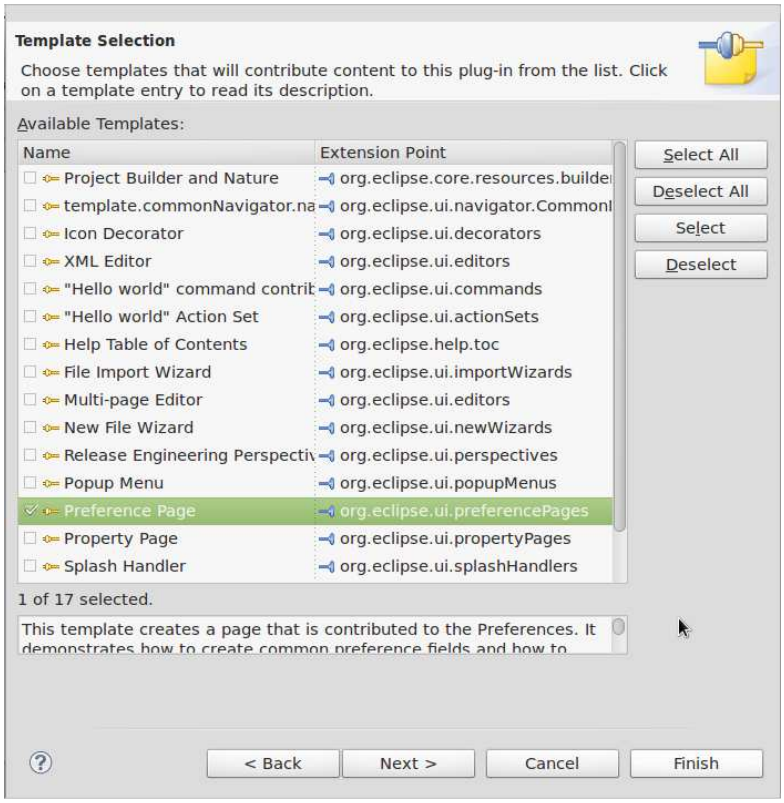


Figura 37. Eclipse PDE (V)

De esta forma ya se encuentra el esqueleto del plugin creado para introducir la funcionalidad deseada como se ha explicado en capítulos anteriores del documento.

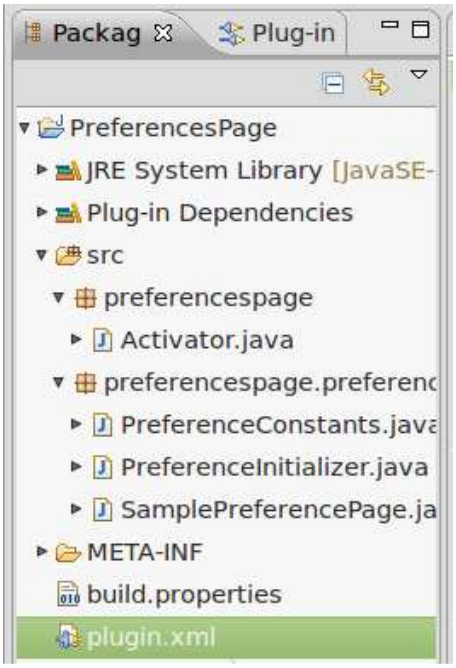


Figura 38. Eclipse PDE (VI)

Se dispone de la clase Activator, maneja la ejecución del plugin, el archivo plugin.xml, controla la ejecución de las distintas clases, y las clases PreferenceConstants, PreferenceInitializer y SamplePreferencePage que es donde se debe incluir la funcionalidad del plugin.

Para probar la funcionalidad del plugin se debe hacer clic con el botón derecho del ratón sobre el nombre del proyecto, seleccionar la opción Run As y después Eclipse Application. De esta manera se ejecutará un nuevo Eclipse con la funcionalidad del plugin incluida.

Anexo 2

Glosario

Refactorización: a refactorización es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

Plugin: Un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Intel TBB: Intel Threading Building Blocks (Intel TBB) es una biblioteca basada en plantillas para C++ desarrollada por Intel para facilitar la escritura de programas que exploten las capacidades de paralelismo de los procesadores con arquitectura multinúcleo.

C++: es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido.

Paralelización automático: Paralelización automática, también auto paralelización, autoparalelización, o paralelización, se refiere a convertir código secuencial en multihilo o vectorizado (o los dos) con objeto de utilizar múltiples procesadores simultaneamente en una máquina con multiprocesador (SMP) de memoria compartida.

Microsoft Office: es una suite de oficina que abarca e interrelaciona aplicaciones de escritorio, servidores y servicios para los sistemas operativos Microsoft Windows y MacOS.

Microsoft Windows: Microsoft Windows es el nombre de una familia de sistemas operativos desarrollados por Microsoft desde 1981, año en que el proyecto se denominaba “Interface Manager”. Las versiones más recientes de Windows son Windows 8 para equipos de escritorio, Windows Server 2011 para servidores y Windows Phone para dispositivos móviles.

Task stealing: Intel TBB implenta task stealing (robo de tareas) para balancear la carga de trabajo sobre los núcleos de procesamiento disponibles con el fin de incrementar el aprovechamiento de los núcleos y la escalabilidad de los programas.

OpenMP: es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas.

Linux: Linux es un núcleo libre de sistema operativo basado en Unix. Es uno de los principales ejemplos de software libre. Linux está licenciado bajo la GPL v2 y está desarrollado por colaboradores de todo el mundo.

Eclipse PDE: Entorno de desarrollo de plugins de Eclipse (Plug-in Development Environment).

Eclipse CDT: Entorno de desarrollo integrado de código abierto multiplataforma para el desarrollo de aplicaciones C/C++ (Eclipse C/C++ Development Tools).

STL: Biblioteca estándar de plantillas (Standard Template Library).

Facebook: es un sitio web de redes sociales creado por Mark Zuckerberg y fundado por Eduardo Saverin, Chris Hughes, Dustin Moskovitz y Mark Zuckerberg. Originalmente era un sitio para estudiantes de la Universidad de Harvard, pero actualmente está abierto a cualquier persona que tenga una cuenta de correo electrónico. Los usuarios pueden participar en una o más redes sociales, en relación con su situación académica, su lugar de trabajo o región geográfica.

g++: es el alias tradicional de GNU C++, un conjunto gratuito de compiladores de C++.

HipHop for PHP: complemento creado por Facebook que permite convertir código PHP en código C++, para posteriormente ser compilado con g++ para obtener así un código objeto mucho más eficiente en consumo de recursos de lo que era el código PHP original.

Java: es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de la sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Parallel_for: Función contenida en la biblioteca Intel TBB para paralelizar bucles for.

Bucle for: El bucle for o ciclo for es una estructura de control en la que se puede indicar el número mínimo de iteraciones.

Runtime: Se denomina tiempo de ejecución al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.

SDK: Un kit de desarrollo de software o SDK es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto (software development kit).

XML: XML, ("eXtensible Markup Language"), es un lenguaje desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Métrica v3: es una metodología de planificación, desarrollo y mantenimiento de sistemas de información, promovida por el Ministerio de Administraciones Públicas de España para sistemizar las actividades del ciclo de vida de los proyectos software dentro de las Administraciones Públicas.

Diagrama de Gantt: El diagrama de Gantt es una popular herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

RadioButton: botón de opción (o radio button) es un elemento de interacción de la interfaz gráfica de una aplicación con el usuario, que permite a éste hacer una única selección de un conjunto de opciones.

Diagrama de flujo: El diagrama de flujo es la representación gráfica del proceso que se va a desarrollar.

Manifest File: En la plataforma Java un archivo manifest es un archivo específico contenido en un archivo Jar. Se usa para definir datos relativos a la extensión y al paquete.

Árbol AST: es una representación de árbol de la estructura sintáctica abstracta (simplificada) del código fuente escrito en cierto lenguaje de programación.

Anexo 3

Referencias

[1] Fundamentos de Paralelización. (s.f.). Obtenido de http://telematica.cicese.mx/computo/super/cicese2000/paralelo/Part2.html

[2] Paralelización automática. (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Paralelizaci%C3%B3n_autom%C3%A1tica

[3] Primeros pasos con Threading Building Blocks. (s.f.). Obtenido de http://jjcoellov.es/2011/01/24/primeros-pasos-con-threading-building-blocks/

[4] Aprenda C++ Básico. (s.f.). Obtenido de http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/cpp/basico/cppbasico.pdf

[5] C++. (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/C%2B%2B

[6] Eclipse (Software). (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Eclipse_%28software%29#Caracter.C3.ADsticas

[7] OpenMP. (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/OpenMP

[8] Intel Threading Building Blocks. (s.f.). Obtenido de Universidad de Murcia: http://dis.um.es/~domingo/apuntes/AlgProPar/1112/tbb.pdf

[9] Intel Threading Building Blocks For Open Source. (s.f.). Obtenido de http://threadingbuildingblocks.org/documentation.php

[10] Intel Threading Building Blocks. (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Intel_Threading_Building_Blocks

[11] Algoritmos paralelos. (s.f.). Obtenido de MSDN: http://msdn.microsoft.com/es-es/library/dd470426.aspx

[12] Introduction to Eclipse Plugin Development. (s.f.). Obtenido de http://www.eclipsepluginsite.com/

[13] Definición de Casos de Uso. (s.f.). Obtenido de http://www.mastermagazine.info/termino/4184.php

[14] UML: Casos de Uso. (s.f.). Obtenido de http://www.ingenierosoftware.com/analisisydiseno/casosdeuso.php

[15] Diagrama de flujo. (s.f.). Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Diagrama_de_flujo

[16] Como desarrollar un plugin para Eclipse. (s.f.). Obtenido de http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=develop-eclipse-plugin